System Description

AC500

Scalable PLC
for Individual Automation

System Technology

AC500 System Technology

CPUs

Interrupt and Counter Module DC541-CM

Ethernet Couplers

PROFIBUS DP Couplers

CANopen Couplers

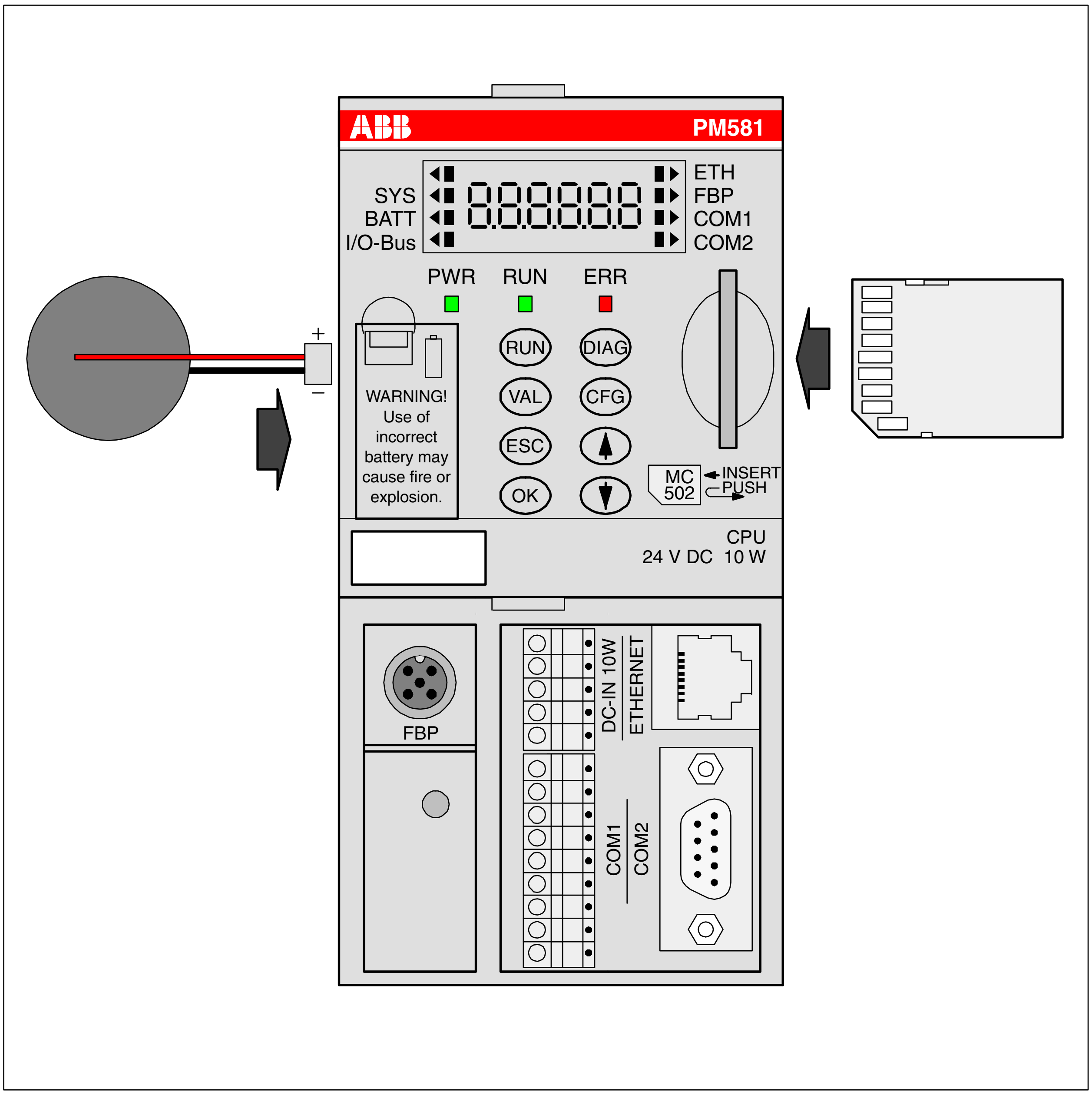
DeviceNet Couplers

System Description

AC500

Scalable PLC
for Individual Automation

System Technology
of the CPUs



Contents - System Technology of the AC500 CPUs

1 Target Support Package

1.1 Introduction

1.1.1 Control Builder PS501 versions

The AC500 basic units PM57x, PM58x and PM59x are programmed using the AC500 Control Builder (version V1.0 and later).

The Control Builder versions V1.0 and 1.1 are based on CoDeSys version V2.3 SP4 Patch 9 (V2.3.4.9+) and later. The Control Builder version V1.2 is based on CoDeSys version V2.3 SP8 Patch 0 (V2.3.8.0) and later.

Note: This documentation applies to all categories of the basic units PM57x, PM58x and PM59x. When the term "PM581" is given in the text, this text applies also to PM57x, PM58x and PM59x. Texts that are exclusively applicable for PM581 are expressly mentioned by a note.

To be able to program the AC500 controllers with the Control Builder, a so-called Target Support Package (TSP) must be installed. By default, the AC500 TSPs are automatically installed during installation of the Control Builder.

The default installation paths are as follows:

- **Control Builder:**
  ..\%ProgramFiles%\3S Software\CoDeSys V2.3
  The environment variable %ProgramFiles% points to the directory "Program Files" on English operating systems and "Programme" on German operating systems.

- **TSP (Target Support Package):**
  ..\%CommonProgramFiles%\CAA-Targets\ABB_AC500
  The environment variable %CommonProgramFiles% points to the directory "Program Files\Common Files" on English operating systems and "Programme\Gemeinsame Dateien" on German operating systems.

The chapter "Installation of a Target Support Package" describes in detail how to install a TSP.

A Target Support Package (TSP) contains all configuration and expansion files necessary to operate a particular controller (target system) with an application. What has to be configured:

- code generator
- memory layout
- controller functions
- I/O modules
- interface assignment and parameterization

In addition, libraries, gateway drivers, target system specific help files, controller configuration files, error description file (Errors.ini) and the help file for the PLC browser (Browser.ini) are included.

Note: The basic units of the AC31 series 90 (07 KT 95, 07 KT 96, 07 KT 97 and 07 KT 98) are still programmed using the programming system 907 AC 1131.

1.1.2 New functions in PS501 V1.2

Version 1.2 implements the following new functions compared to versions V1.0 and V1.1:

- CD user interface in German, English, French and Spanish
- New CoDeSys version with display of the ABB Control Builder version. Available menu languages: German, English, French, Spanish, Italian and Russian
- Allows the installation of the CoDeSys HMI
- New AC500 targets PM5xy_V12 for downward compatibility of projects created with versions V1.0/V1.1
- ABB EDS configurator for the generation of fix configurations from modular EDS files for DeviceNet
- Revised and expanded documentation
- Ethernet UDP: Selection of the UDP port in the PLC Configuration
- Ethernet/Online: Driver "ABB Tcp/Ip AC"
  (AA/55 protocol, same driver for AC500 and AC31 S90/07KT9x)
- COMx: Protocol "SysLibCom" - Support of the blocks contained in the 3S library SysLibCom.lib
- COMx: Protocol "Multi" - Switching between two protocols, for example ASCII/Modbus with the FB COM_SET_PROT
- COMx: Parameter 'Enable login' for CoDeSys login via COMx, if COMx is configured with the protocol "Modbus", "ASCII", "SysLibCom" or "Multi"
- Integration of the CS31 device DC551-CS31
- ARCNET online access using the driver "ABB ARCNET AC"
- ARCNET-_5F_ARC - 5F-ARCNET protocol (as in 07KT97/07KT98)
- ARCNET data exchange with the FBs ARC_SEND, ARC_REC, ARC_STO
- ARCNET-FB: ARC_INFO, ARC_OWN_NODE
- SD card: Firmware download for field bus couplers
- New PERSISTENT area "%R area", configurable as VAR RETAIN PERSISTENT in the PLC Configuration / CPU parameters
- PLC - Browser: Command "coupler settings" to display, for example, the IP address and the socket assignment of the Ethernet coupler
- Call stack in case of an application crash
- CPU parameters: 'Warmstart'->off/E2/voltage dip/e2 or voltage dip
  (automatic restart of the CPU after E2 errors and/or short voltage dips, as in 07KT98)
- STOP of the user program if a task has been suspended (all outputs set to FALSE/0)
- FPU-Exception: FB FPUEXINFO (for PM591, PM590 only)
- New CPU: PM590
- Blocks that enable the export/import of RETAIN data to/from the SD card (except %M area)
- Blocks that enable the export/import of the PERSISTENT area (%R area) to/from the SD card

Note: The new functions implemented in version V1.2 can only be used together with the targets PM5xy_V12, CoDeSys as of version V2.3.8.0 and the AC500 firmware as of version V1.2.0. Further information about this can be found in the chapter "Compatibility of versions V1.0, V1.1 and V1.2".

1.1.3 Compatibility of versions V1.0, V1.1 and V1.2

The new functions in version 1.2 listed in the chapter above require extensive changes in the firmware of the AC500 CPUs, in the target files and also according changes in CoDeSys.

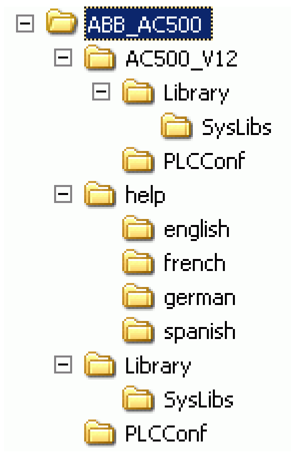
To ensure downward compatibility with versions V1.0 and V1.1, new target system files for all AC500 CPUs have been created for version V1.2. These files are installed in parallel to the target system files of version V1.1 when installing Control Builder V1.2. Only the newest online help files will be installed.

File structure of the target system

The target system files are structured as follows in version V1.2:

Installation path (as for V1.0 and V1.1):
..\%CommonProgramFiles%\CAA-Targets\ABB_AC500

The environment variable %CommonProgramFiles% is points to the directory
- "Program Files\Common Files" on English operating systems and
- "Programme\Gemeinsame Dateien" on German operating systems.



Overview on target system files

The following files are part of the target:

No.	File / Directory	V1.0	V1.1	V1.2 see remark 1
1. Target files		Root = ..\		Root = ..\
1	AC500_PM571.trg	2005-07-28	2005-07-28	-
2	AC500_PM581.trg	2005-07-28	2005-07-28	-
3	AC500_PM591.trg	2005-07-28	2005-07-28	-
4	AC500_multi.tnf (install)	2005-06-01	2005-06-01	-
5	AC500_PM582.trg	-	2005-08-23	-
6	AC500_PM582.tnf (install)	-	2005-08-23	-
7	AC500_PM571_V12.trg	-	-	2007-05-16
8	AC500_PM581_V12.trg	-	-	2007-05-16
9	AC500_PM582_V12.trg	-	-	2007-05-16
10	AC500_PM590_V12.trg	-	-	2007-05-16
11	AC500_PM591_V12.trg	-	-	2007-05-16
12	AC500_V12.tnf (install)	-	-	2007-05-23

Remark 1:
The file date of the version V1.2 files is the date set on the CD PS501 V1.2 BETA 1. Thus, the release version may have other files.

No.	File / Directory	V1.0	V1.1	V1.2 see remark 1
2. Target ancillary files		**Root = ..**		**Root = ..\AC500_V12**
1	AC500.hll	2004-05-27	2004-05-27	2007-04-26
2	Browser.ini	2005-09-07	2006-02-21	2006-06-02
3	Errors.ini	2005-05-23	2006-05-23	2006-05-23
4	Errors.xml (in CoDeSys directory)	2005-06-13	2006-05-29	2006-05-29
5	TaskConfig.xml	2004-04-21	2006-02-21	-
6	TaskConfig_PM57x.xml	-	-	2006-02-21
7	TaskConfig_PM58x.xml	-	-	2006-02-21
8	TaskConfig_PM59x.xml	-	-	2006-02-21

Remark 1:
The file date of the version V1.2 files is the date set on the CD PS501 V1.2 BETA 1. Thus, the release version may have other files.

No.	File / Directory	V1.0	V1.1	V1.2 see remark 1
3. Configuration files see remark 2		**Root = ..\PLCConf**		**Root = ..\AC500_V12\PLCConf**
1	AC500_COMx_V10.cfg	2005-07-26	2006-07-20	2007-04-19
2	AC500_CPU_V10.cfg	2005-07-26	2006-05-23	2007-04-19
3	AC500_FBPSlave_V10.cfg	2005-07-26	2006-05-23	2007-04-11
4	AC500_ModCS31_V10.cfg	2005-07-26	2006-07-24	2007-04-11
5	AC500_IOmodule_V10.cfg	2005-09-06	2006-07-28	2007-04-24
6	AC500_Coupler_V10.cfg	2005-09-07	2006-05-23	2007-04-13
7	AC500_CS31Base_V10.cfg	-	2006-05-23	2007-04-11
8	AC500_DC541_V11.cfg	-	2006-05-25	2007-04-13
9	AC500_IOmod2_V11.cfg	-	2006-07-04	2007-04-11
10	AC500_CAN_DevNet_V11.cfg	-	2006-07-13	2007-04-11
11	AC500_IOclass_V12.cfg	-	-	2007-04-11
12	AC500_ARCNET_V12.cfg	-	-	2007-04-11
13	AC500_COMNewProt_V12.cfg	-	-	2007-04-19

Remark 1:
The file date of the version V1.2 files is the date set on the CD PS501 V1.2 BETA 1. Thus, the release version may have other files.

Remark 2:
The directory ..\PLCConfig may only contain the *.cfg files listed here because all *.cfg files are loaded when loading a project. If the directory contains, for example, configuration files with partly the same content, both files are loaded. This can lead to unforeseeable effects!

No.	File / Directory	V1.0	V1.1	V1.2 see remark 1
4. Libraries		**Root = ..\Library**		**Root = ..\AC500_V12\Library**
1	BusDiag-lib	2004-08-27	2004-08-27	2004-08-27
2	CS31_AC500_V10.lib	2005-06-01	2005-06-01	2005-06-01
3	SysExt_AC500_V10.lib	2005-07-01	2005-07-01	2005-07-01
4	Serie90_AC500_V10.lib	2005-07-27	2005-07-27	2005-07-27
5	ASCII_AC500_V10.lib	2005-07-27	2005-07-27	2006-11-08
6	MODBUS_AC500_V10.lib	2005-07-28	2006-06-01	2006-06-01
7	Ethernet_AC500_V10.lib	2005-07-28	2005-07-28	2005-07-28
8	SysInt_AC500_V10.lib	2005-08-03	2006-06-02	2007-05-24
9	Diag_AC500_V10.lib	2005-08-10	2005-08-10	2005-08-10
10	PROFIBUS_AC500_V10.lib	2005-08-12	2005-08-12	2005-08-12
11	DeviceNet_AC500_V11.lib	-	2006-01-26	2006-01-26
12	CANopen_AC500_V11.lib	-	2006-03-02	2006-03-02
13	DC541_AC500_V11.lib	-	2006-06-01	2007-04-13
14	Counter_AC500_V11.lib	-	2006-06-02	2006-06-02
15	ARCNET_AC500_V12.lib	-	-	2007-03-01

Remark 1:
The file date of the version V1.2 files is the date set on the CD PS501 V1.2 BETA 1. Thus, the release version may have other files.

No.	File / Directory	V1.0	V1.1	V1.2 see remark 1
5. System libraries (partly) supported by AC500		**Root = ..\Library\SysLibs**		**Root = ..\AC500_V12\ Library\SysLibs**
1	SysLibCallback.lib	2005-07-18	2005-07-18	2005-07-18
2	SysLibCom.lib	2005-07-18	2005-07-18	2005-07-18
3	SysLibEvent.lib	2005-07-18	2005-07-18	2005-07-18
4	SysLibIecTasks.lib	2005-07-18	2005-07-18	2005-07-18
5	SysLibMem.lib	2005-07-18	2005-07-18	2005-07-18
6	SysLibPLCConfig.lib	2005-07-18	2005-07-18	2005-07-18
7	SysLibPlcCtrl.lib	2005-07-18	2005-07-18	2005-07-18
8	SysLibProjectInfo.lib	2005-07-18	2005-07-18	2005-07-18
9	SysLibRtc.lib	2005-07-18	2005-07-18	2005-07-18
10	SysLibSem.lib	2005-07-18	2005-07-18	2005-07-18
11	SysLibStr.lib	2005-07-18	2005-07-18	2005-07-18
12	SysLibTasks.lib	2005-07-18	2005-07-18	2005-07-18
13	SysLibTime.lib	2005-07-18	2005-07-18	2005-07-18
14	SysLibVisu.lib	-	-	2002-08-08
15	SysTaskInfo.lib	2005-07-18	2005-07-18	2005-07-18

Remark 1:
The file date of the version V1.2 files is the date set on the CD PS501 V1.2 BETA 1. Thus, the release version may have other files.

No.	File / Directory	V1.0	V1.1	V1.2 see remark 1
6. Online help German version		**Root = ..\help\german**		
1	CAA-Merger-1.chm	2005-09-30	2006-08-04	2006-08-04
2	CAA-Merger-2.chm	2006-04-07	2006-07-06	2007-05-22
3	CAA-Merger-3.chm	2005-08-11	2005-08-11	2005-08-11
4	CAA-Merger-6.chm	2006-03-27	2006-06-29	2007-05-14
5	CAA-Merger-7.chm	2006-03-16	2006-09-27	2006-09-27
6	CAA-Merger-8.chm	2005-09-14	2005-09-14	2005-09-14
7. Online help English version		**Root = ..\help\english**		
1	CAA-Merger-1.chm	2005-09-30	2006-08-04	2006-08-04
2	CAA-Merger-2.chm	2006-04-12	2006-07-06	2007-05-22
3	CAA-Merger-3.chm	2005-08-18	2005-08-18	2005-08-11
4	CAA-Merger-6.chm	2006-03-27	2006-06-29	2007-05-14
5	CAA-Merger-7.chm	2006-04-18	2006-10-04	2006-09-27
6	CAA-Merger-8.chm	2005-09-14	2005-09-14	2005-09-14

Remark 1:
The file date of the version V1.2 files is the date set on the CD PS501 V1.2 BETA 1. Thus, the release version may have other files.

No.	File / Directory	V1.0	V1.1	V1.2 see remark 1
8. Online help French version		**Root = ..\help\french (see remark 3)**		
1	CAA-Merger-1.chm	-	2006-08-04	2006-08-04
2	CAA-Merger-2.chm	-	2006-07-06	2007-05-22
3	CAA-Merger-3.chm	-	2005-08-18	2005-08-11
4	CAA-Merger-6.chm	-	2006-06-29	2007-05-14
5	CAA-Merger-7.chm	-	2006-10-04	2006-09-27
6	CAA-Merger-8.chm	-	2005-09-14	2005-09-14
9. Online help Spanish version		**Root = ..\help\spanish (see remark 3)**		
1	CAA-Merger-1.chm	-	-	2006-08-04
2	CAA-Merger-2.chm	-	-	2007-05-22
3	CAA-Merger-3.chm	-	-	2005-08-11
4	CAA-Merger-6.chm	-	-	2007-05-14
5	CAA-Merger-7.chm	-	-	2006-09-27
6	CAA-Merger-8.chm	-	-	2005-09-14

Remark 1:
The file date of the version V1.2 files is the date set on the CD PS501 V1.2 BETA 1. Thus, the release version may have other files.

Remark 3:
The online help is currently only available in German and English. As soon as the translation is completed, the according files will be replaced. The English help files are currently used in the Spanish and French version.

Compatibility of CPU bootcode, CPU firmware, target system and CoDeSys

The following table shows the possible combinations of CPU bootcode, CPU firmware, target system files and CoDeSys.

		Bootcode			Firmware			Target			CoDeSys			
		V1.0.2	V1.1.3	V1.2.0	V1.0.2	V1.1.7	V1.2.0	V1.0	V1.1	V1.2	V2.3.4.9+	V2.3.5.x.. V2.3.7.5	V2.3.7.5+ Test	V2.3.8.0
Bootcode	V1.0.2				++	+	--	++	++	--	++	+	+	++
	V1.1.3				++	++	--	++	++	--	++	+	+	++
	V1.2.0				--	--	++	++	++	++	+	+	+	++
Firmware	V1.0.2	++	++	--				++	+	--	++	+	+	++
	V1.1.7	+	++	--				+	++	--	++	+	+	++
	V1.2.0	--	--	++				+	+	++	+	+	+	++
Target	V1.0	++	++	++	++	+	+				++	+	+	++
	V1.1	++	++	++	+	++	+				++	+	+	++
	V1.2	--	--	++	--	--	++				+	+	+	++
CoDeSys	V2.3.4.9+	++	++	+	++	++	+	++	++	+				
	V2.3.5.x .. V2.3.7.5	+	+	+	+	+	+	+	+	+				
	V2.3.7.5+ Test	+	+	+	+	+	+	+	+	+				
	V2.3.8.0	++	++	++	++	++	++	++	++	++				

++	=> all functions available
+	=> possible, but not all functions available
--	=> combination **NOT** possible

Conversion of a project created with version V1.0 or V1.1 to version V1.2

Proceed as follows to convert a project created with version V1.0 or V1.1 to version V1.2:

1. Safe the project (for example to the directory ..\save_projects)
2. Open the project in CoDeSys
3. Project => Save as (for example under the name name_V12.pro)
4. Resources => Target Settings => Select AC500 PM5xy V1.2
5. Execute Project => Clean all and Project => Rebuild all
6. File => Save
7. Online => Download
8. Online => Create boot project
9. Online => Run

Caution: A project created for the target PM5xy V1.2 can only be loaded into an AC500 PM5xy with bootcode as of version V1.2.0 and firmware as of version V1.2.0.
If a controller with firmware version V1.1.7 is loaded, CoDeSys reports an according error after the download!

If a version V1.2 target is selected, the target system files, libraries and configuration files contained in the directory ..\ABB_AC500\AC500_V12 will be loaded. This ensures that the new functions will be available.

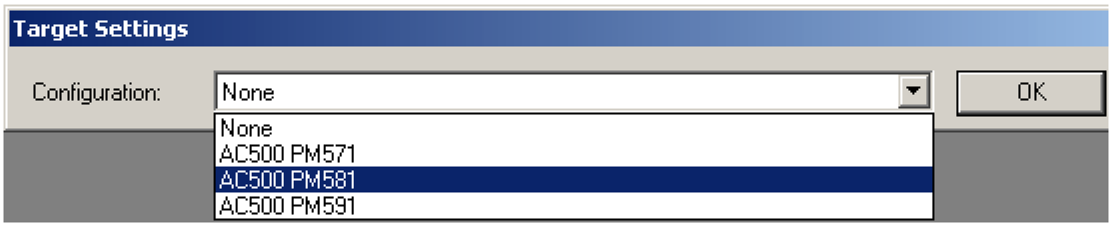
A **change back** to a target of version V1.0 or V1.1 is no longer possible as soon as a parameter, that is only available as of version V1.2, is set in the PLC configuration. For example, if the protocol "Multi" is set for the serial interface or the CPU parameter "Warmstart" is set to the value "On after E2 or short voltage dip". Since the according settings are not available in the configuration files of the targets of version V1.0 or V1.1, the PLC configuration cannot be loaded and has to be recreated, if necessary!

Thus, always save the projects before doing the conversion!

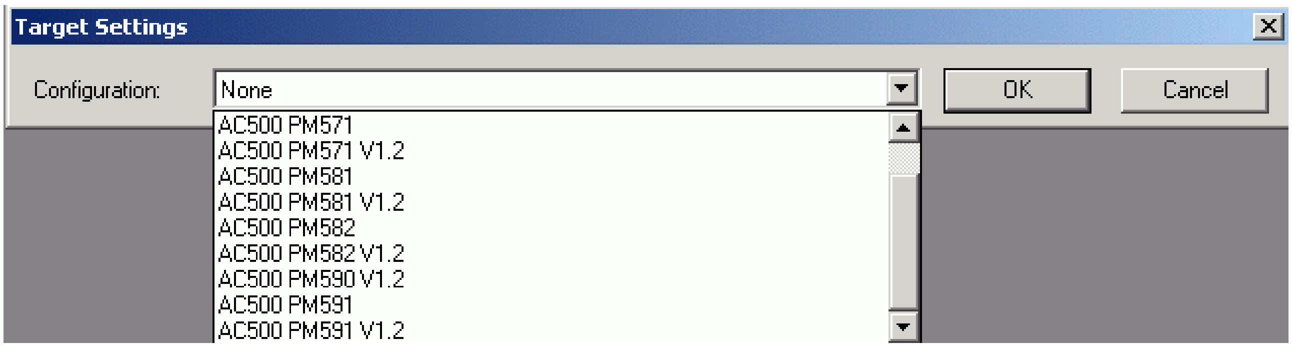
1.2 Selection of the target system - Target support settings

The assignment of a project to an AC500 CPU is done using the target support settings in the Control Builder.

When creating a new AC500 project with File / New, the target system must be selected first:



As of Control Builder version V1.2, the following AC500 targets can be selected.



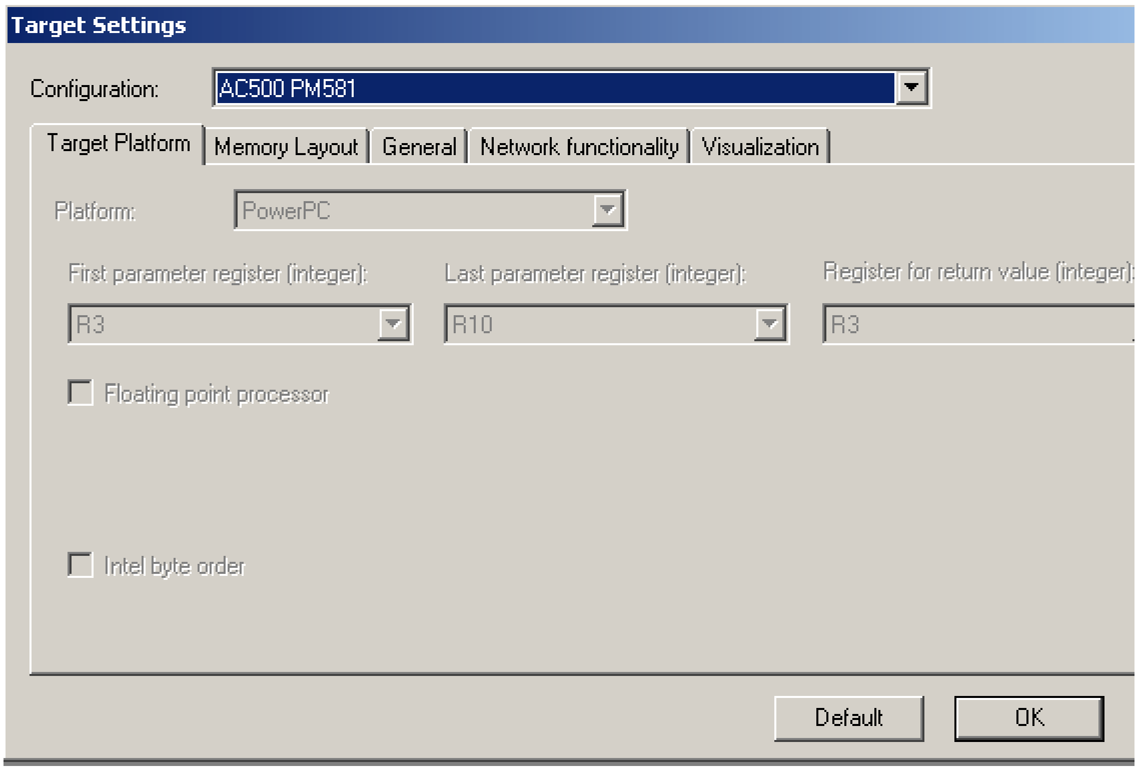
Select the desired CPU and confirm your selection with OK.

Note: If you want to download an existing project to another AC500 CPU, the new CPU must be set in the target system settings. To do this, select the object "Target Settings" in the "Resources" tab. In the appearing window, select the desired CPU from the "Configuration" list box.

1.3 CPU parameters in the target support settings

1.3.1 "Target Platform" settings

Once you have selected an AC500 CPU, the general CPU parameters are displayed:



Configuration: AC500 PM581

Platform: PowerPC

First parameter register (integer): R3

Last parameter register (integer): R10

Register for return value (integer): R3

☐ Floating point processor

☐ Intel byte order

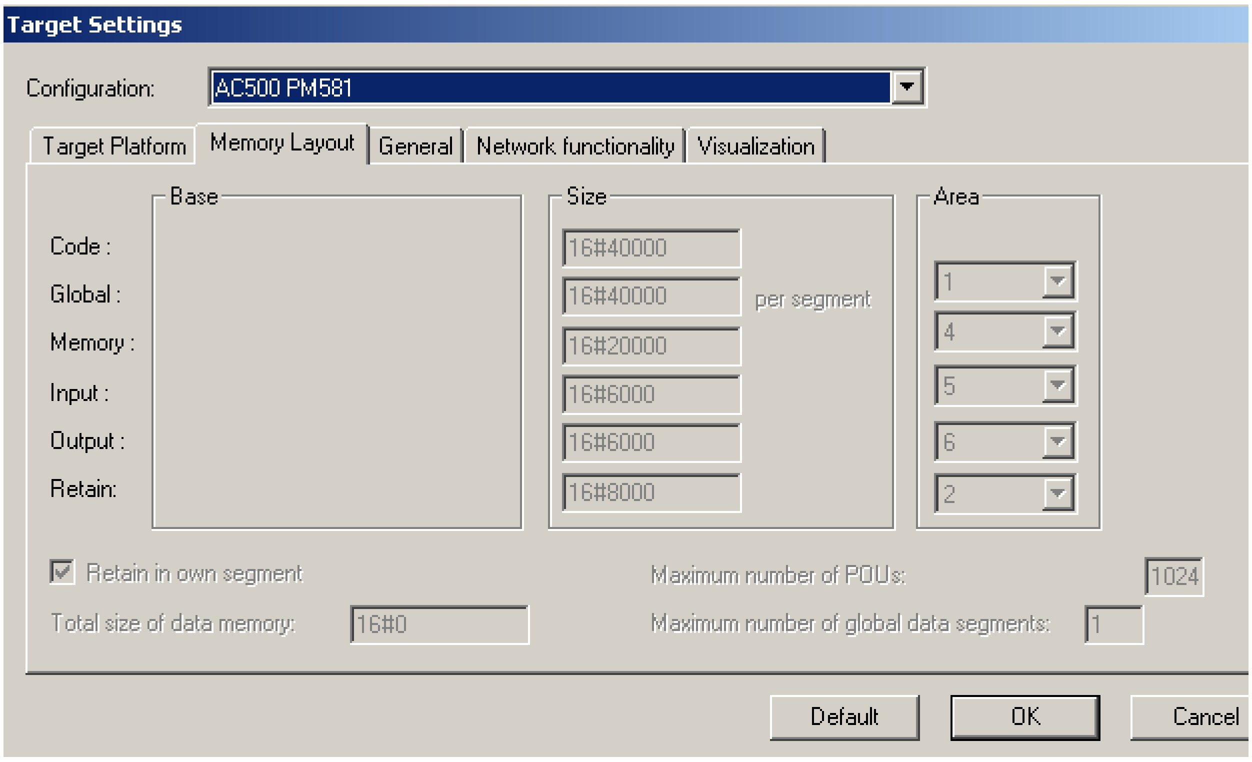
The tab "Target Platform" displays the general settings for the CPU. These parameters cannot be changed.

The parameters have the following meaning:

Parameter	Meaning
Platform	Target system type - PowerPC
Floating point processor	PM57x/PM58x: Floating point operations are emulated PM59x: Floating point processor
First parameter register (integer)	Register where the first (integer) parameter of C function calls is passed (range depends on the operating system).
Last parameter register (integer)	Register where the last (integer) parameter of C function calls is passed (range depends on the operating system).
Register for return value (integer)	Register where the integer values of C function calls are returned (range depends on the operating system).
Intel byte order	If deactivated: Motorola byte address scheme is applied.

1.3.2 "Memory Layout" settings

The tab "Memory Layout" contains all information about the memory mapping within the CPU:



Target Settings

Configuration: AC500 PM581

Target Platform | Memory Layout | General | Network functionality | Visualization

	Base	Size		Area
Code :		16#40000		1
Global :		16#40000	per segment	4
Memory :		16#20000		5
Input :		16#6000		6
Output :		16#6000		6
Retain:		16#8000		2

☑ Retain in own segment

Maximum number of POUs: 1024

Total size of data memory: 16#0

Maximum number of global data segments: 1

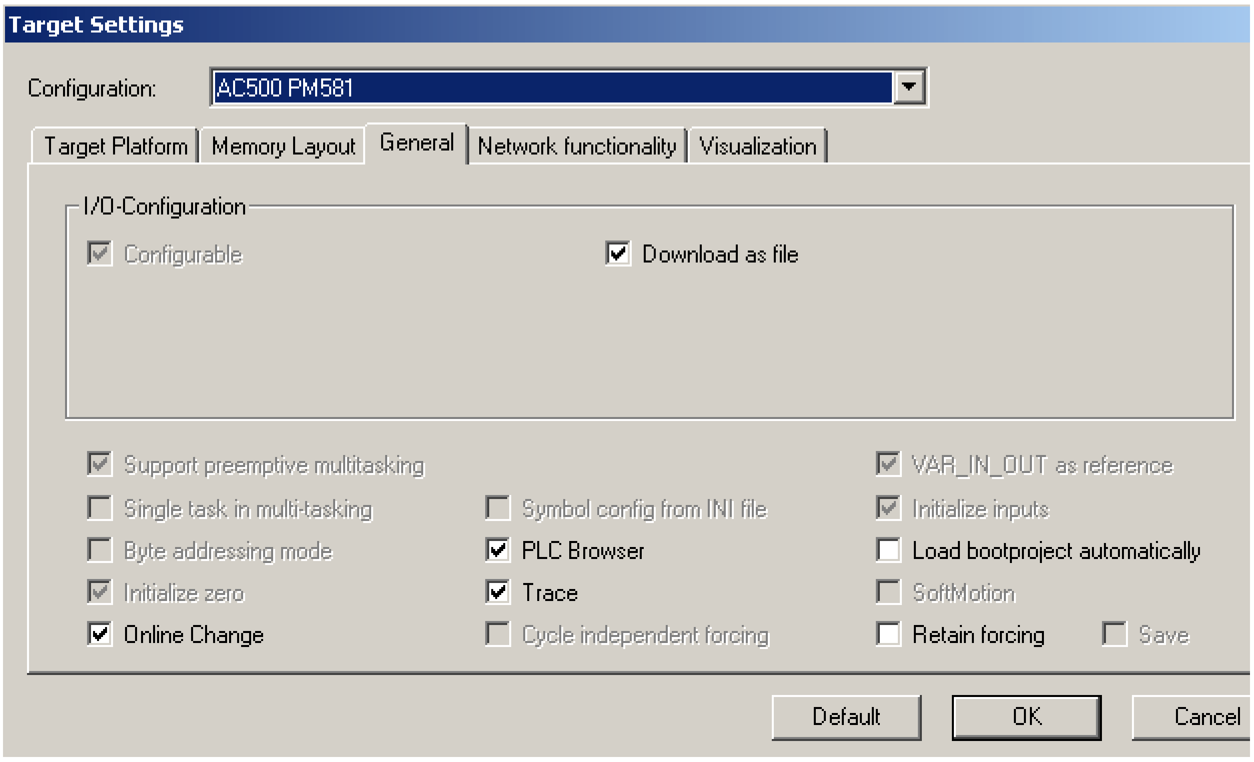
The parameters have the following meaning:

Parameter	Meaning	
Size (Code)	Code area size (user program) see chapter "User program size and operands of AC500 CPUs"	
Size per segment (Global)	Data area size (VAR ... END_VAR or VAR_GLOBAL & END_VAR) see chapter "User program size and operands of AC500 CPUs"	
Area (Memory)	Size of the addressable flag area (%M area) see chapter "User program size and operands of AC500 CPUs"	
Area (Input)	Size of the input process image (%I area) PM5x1: 6000_{hex} = 24 kB	
Area (Output)	Size of the output process image (%Q area) PM5x1: 6000_{hex} = 24 kB	
Size (Retain)	Size of the area for retentive data (VAR RETAIN) see chapter "User program size and operands of AC500 CPUs"	
Retain in own segment	If activated: Retentive data are administered in an own segment	
Total size of data memory	Can be ignored, set in firmware	
Maximum number of POUs	Maximum number of blocks allowed in the project see chapter "User program size and operands of AC500 CPUs"	
	The following applies:	1 POU
	- per function	1 POU
	- per program	2 POUs
	- per function block	1 POU
	- per data type	
	For library POUs the same number of POUs for the relevant type is valid.	
Maximum number of global data segments	1 = all global data are administered in one segment.	

The numbers for the individual areas are required for administration in the firmware. The user cannot change any of these parameters.

1.3.3 "General" settings

The tab "General" contains general information about the target system and the relevant windows in the Control Builder.


Target Settings

Configuration: AC500 PM581

I/O-Configuration

- [✓] Configurable
- [✓] Download as file

- [✓] Support preemptive multitasking
- [ ] Single task in multi-tasking
- [ ] Byte addressing mode
- [✓] Initialize zero
- [✓] Online Change
- [ ] Symbol config from INI file
- [✓] PLC Browser
- [✓] Trace
- [ ] Cycle independent forcing
- [✓] VAR_IN_OUT as reference
- [✓] Initialize inputs
- [ ] Load bootproject automatically
- [ ] SoftMotion
- [ ] Retain forcing
- [ ] Save

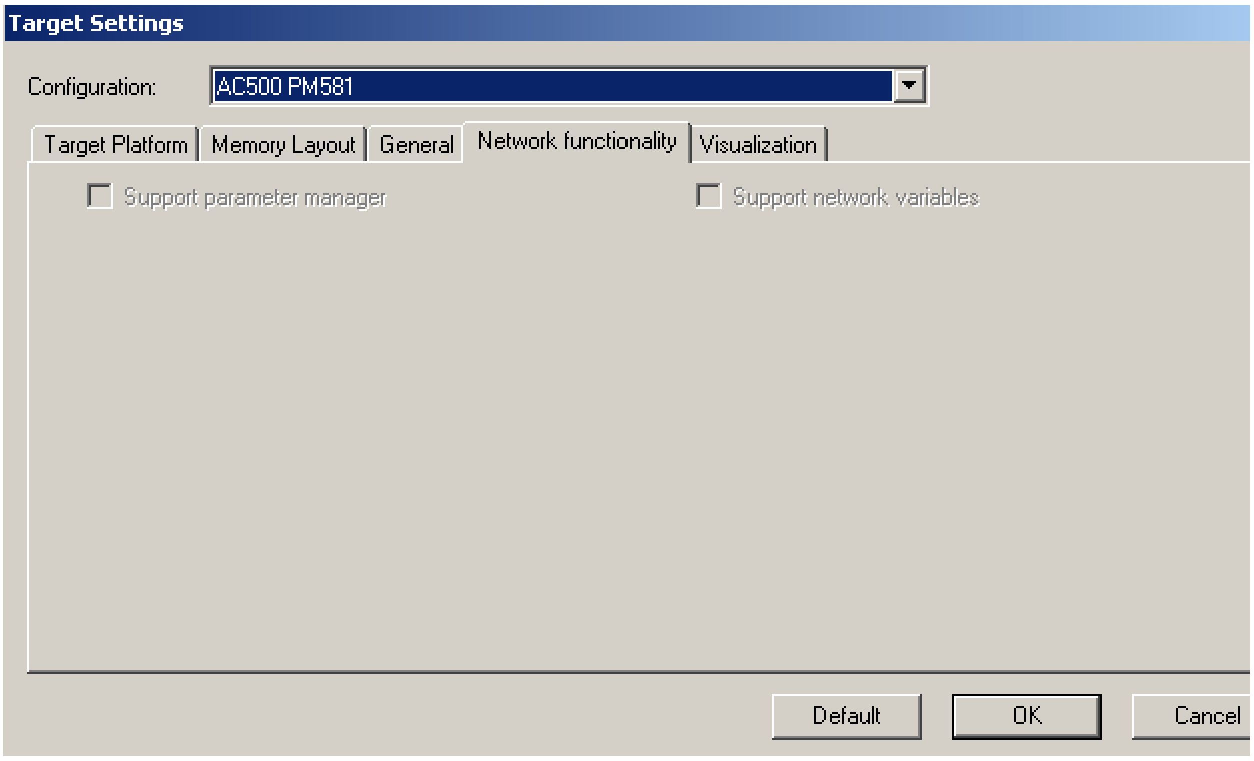
The parameters have the following meaning:

Parameter	Meaning
Configurable	**If activated:** Support configurable I/O configurations and download configuration descriptions to the controller
Download as file	**If activated:** When downloading, the I/O configuration file is downloaded to the controller together with the project. If the checkbox is **deactivated**, the I/O configuration will be downloaded to the controller in the download format **without displaying the download progress**.
Support preemptive multitasking	**If activated:** Support task configuration and download task description to controller
No address checking	**If deactivated:** When compiling the project, the IEC addresses are checked
Online Change	**If activated:** Online Change functionality
Singletask in multitasking	not yet implemented
Byte addressing mode	**If deactivated:** Byte addressing mode
Initialize zero	**If activated:** General initialization with zero
Download symbol file	**If deactivated:** The symbol file possibly created during the download will not be loaded into the controller.
Symbol config from INI file	**If deactivated:** The parameters for the symbol configuration are read from the project options window.
PLC Browser	**If activated:** PLC browser functionality supported.
Trace	**If activated:** Trace recording possible.
Cycle independent forcing	not yet implemented
VAR_IN_OUT as reference	**If activated:** At a function call, VAR_IN_OUT variables are called by reference (pointer); therefore no constants can be assigned and no read/write access is possible from outside the function block.
Initialize inputs	**If activated:** Init code is generated for the inputs declared with "AT %IX".
Load bootproject automatically	**If activated:** After the download, a bootproject is created automatically from the new program and sent to the PLC.
SoftMotion	**If deactivated:** No SoftMotion functionality supported
Retain forcing	**If activated:** Forced values are kept even at logout (provided the user has decided to retain the force list).
Save	If the options "Retain forcing" and "Save" are activated, the force list is stored to flash memory when "Creating the bootproject".

Parameters with gray background can be changed by the user.

1.3.4 "Network Functionality" settings

The tab "Network functionality" contains the settings for the network variables. Network variables are not supported in the current version.


Target Settings

Configuration: AC500 PM581

Target Platform | Memory Layout | General | Network functionality | Visualization

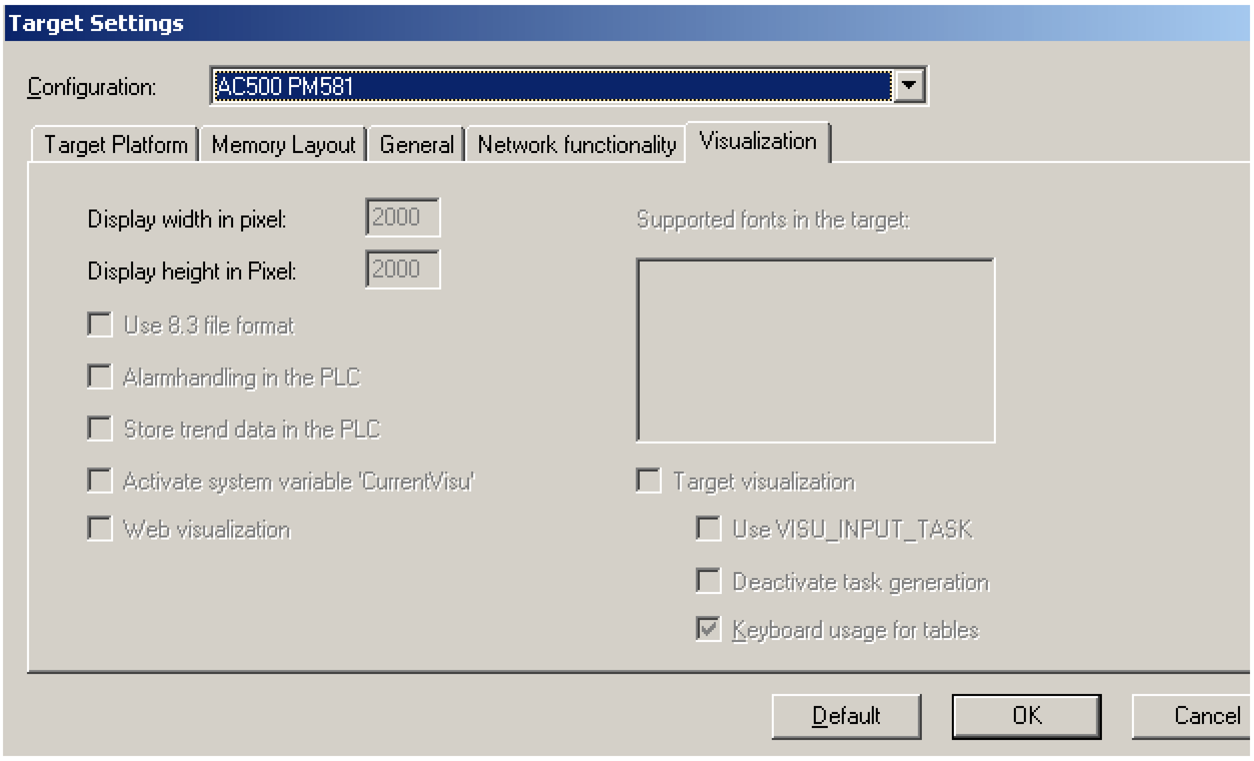
☐ Support parameter manager

☐ Support network variables

Default | OK | Cancel

1.3.5 "Visualization" settings

The tab "Visualization" contains the settings for the visualization integrated in the PLC firmware. Target and WEB visualization are not supported in the current version.


Target Settings

Configuration: AC500 PM581

Target Platform | Memory Layout | General | Network functionality | Visualization

Display width in pixel: 2000

Display height in Pixel: 2000

☐ Use 8.3 file format

☐ Alarmhandling in the PLC

☐ Store trend data in the PLC

☐ Activate system variable 'CurrentVisu'

☐ Web visualization

Supported fonts in the target:

☐ Target visualization

☐ Use VISU_INPUT_TASK

☐ Deactivate task generation

☑ Keyboard usage for tables

Default | OK | Cancel

1.4 Overview on user program size and operands of AC500 CPUs

The following table shows the values set for the program memory and the operands in AC500 targets:

Parameter / CPU	**Unit Version**	**PM571**	**PM581**	**PM582**	**PM590**	**PM591**
Available as of PS501 version		V1.0	V1.0	V1.1	V1.2	V1.1
Available as of firmware version		V1.0.2	V1.0.2	V1.1.7	V1.2.0	V1.1.7
User program (code) **see remark 1**	kB	64	256	512	2048 = 2 MB	4096 = 4 MB
Number of POUs		1024	1024	1024	4096	4096
Number of tasks		3	3	3	16	16
Floating point processor **see remark 2**		no	no	no	yes	yes
Global and local variables: VAR or VAR GLOBAL	kB	16	128	128	1024	2048
RETAIN area: VAR RETAIN or VAR RETAIN PERSISTENT **see remark 3**	kB up to V1.1.x	1	32	32	512	512
	kB V1.2.0 and later	4	32	32	512	512
Addressable flag area: VAR AT %Mx.y VAR RETAIN AT %Mx.y VAR RETAIN PERSISTENT AT %Mx.y	kB	4	128	128	512	512
PERSISTENT area: VAR AT %Rx.y	kB V1.2.0 and later	4	128	128	512	512
Inputs %I **see remark 4**	kB	24	24	24	24	24
Outputs %Q **see remark 4**	kB	24	24	24	24	24

Remark 1:
The user program is composed of:
- the compiled code of all POUs called in the program,
- the initialization code for the variables and
- the code to restore the variables set as PERSISTENT (this does not include variables in the PERSISTENT area (%R area)!).

The configuration data are not considered in the user program size.

As of Control Builder version 1.2, the user program size (code size) is shown in the CoDeSys message box when compiling:

```
POU indexes: 262 (25%)
Size of data used: 1069 of 262144 bytes (0.41%)
Size of Retain data used: 0 of 32768 bytes (0.00%)
Code size: 6726 bytes
0 errors, 0 warnings
```

Remark 2:
All AC500 CPUs can perform floating point operations. For CPUs without floating point processor (PM57x and PM58x), these operations are performed by an emulation library and are therefore slower. Emulation is faster for LREAL variables than for REAL variables. Thus, the use of LREAL variables is recommended.

Remark 3:
The information shown in the message box does only contain the Retain data of the RETAIN area, and not the variables of the addressable flag area %Mx.y. that are declared as VAR RETAIN.

Remark 4:
The assignment of the inputs and outputs is described in detail in the chapter "AC500 inputs, outputs and flags".

1.5 Installation of AC500 targets with the program installTarget.exe

By default, all AC500 target files are installed when installing the Control Builder. When installing version V1.2, the target files of version V1.1 and the files of version V1.2 will be installed.

In certain cases the target files of several providers have to be installed on the PC at the same time. In this case, the target files of the AC500 CPUs can also be installed directly.

Note: The AC500 target of version V1.0 and V1.1 requires CoDeSys version V2.3.5.0 and later. Targets of version V1.2 require CoDeSys version V2.3.8.0 and later. AC500 targets cannot be used with CoDeSys V3.x.

Proceed as follows to install the target:

1. Exit CoDeSys.
2. Exit the gateway, if necessary (right mouse click on the 3S icon in the Windows status bar => Exit).
3. Start the program InstallTarget.exe, for example by double clicking the file in the Windows Explorer. The program InstallTarget.exe is located in the installation directory of CoDeSys.exe, i.e., in case of a default installation in the directory ..\%ProgramFiles%\3S Software\CoDeSys V2.3.
   The environment variable %ProgramFiles% points to the directory
   - "Program Files" on English operating systems and
   - "Programme" on German operating systems.
4. The following user interface appears:

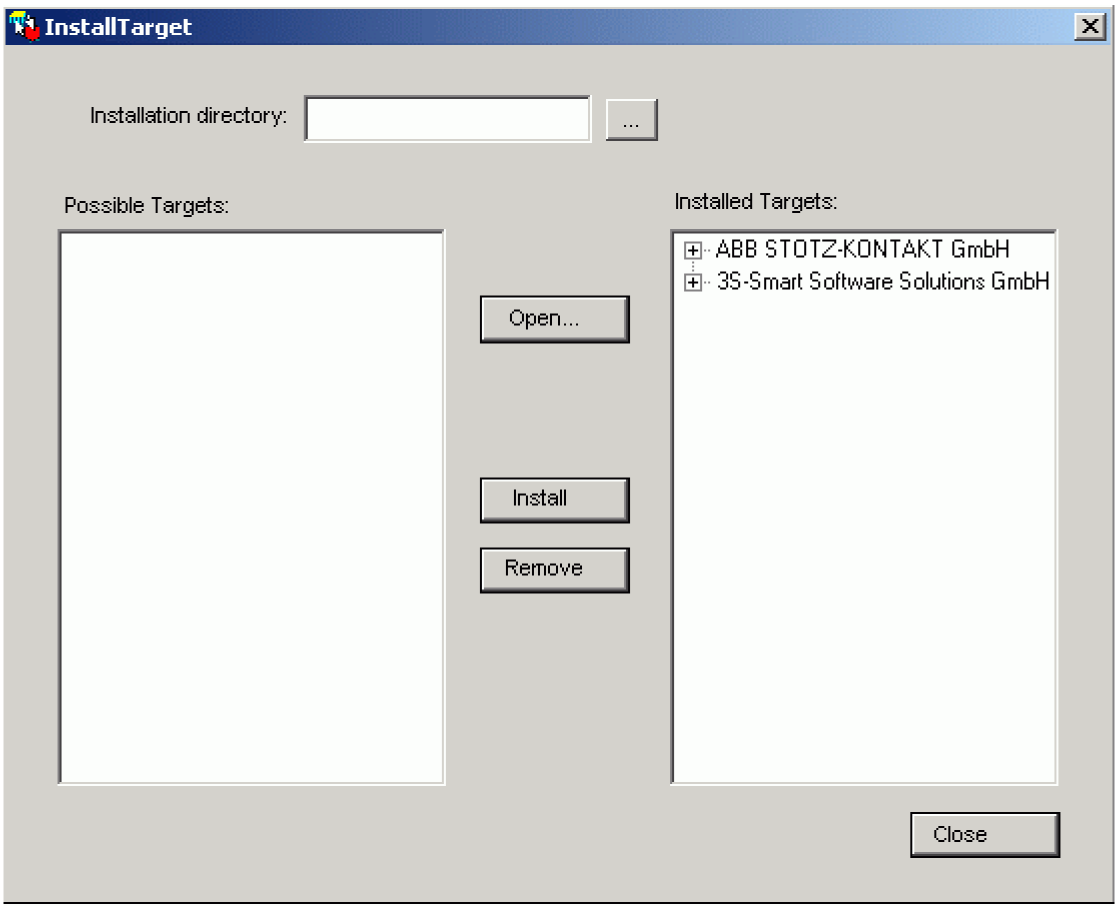
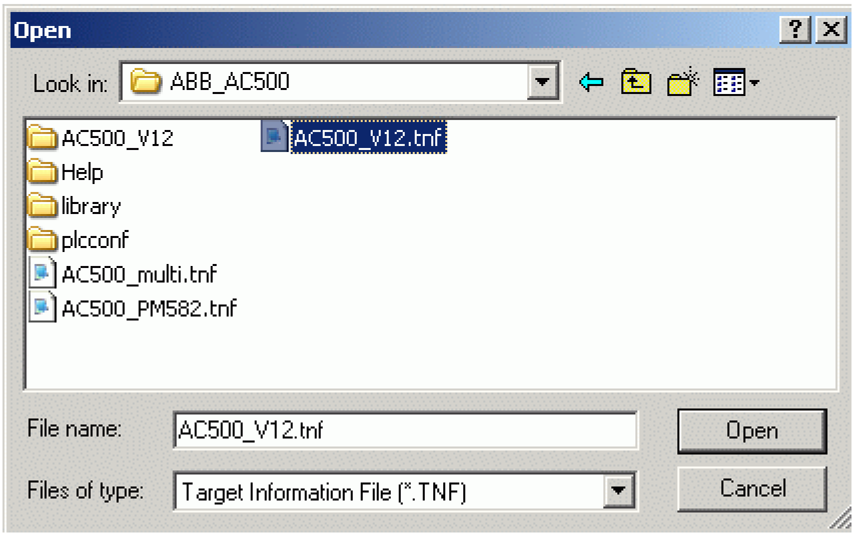


ABB STOTZ-KONTAKT GmbH
3S-Smart Software Solutions GmbH

All targets currently installed (i.e., registered in the Windows registry) are displayed in the right-hand area "Installed targets".

5. Click <Open> and select the directory that contains the installation file of the target(s) to be installed. The installation file of a target has the extension *.tnf. The AC500 targets are located in the directory **CD-ROM drive:\CD_AC500\CoDeSys\Targets\ABB_AC500** on the CD PS501 Vx.y.

6. Select the desired installation file:



Look in: ABB_AC500

AC500_V12
Help
library
plcconf
AC500_multi.tnf
AC500_PM582.tnf
AC500_V12.tnf

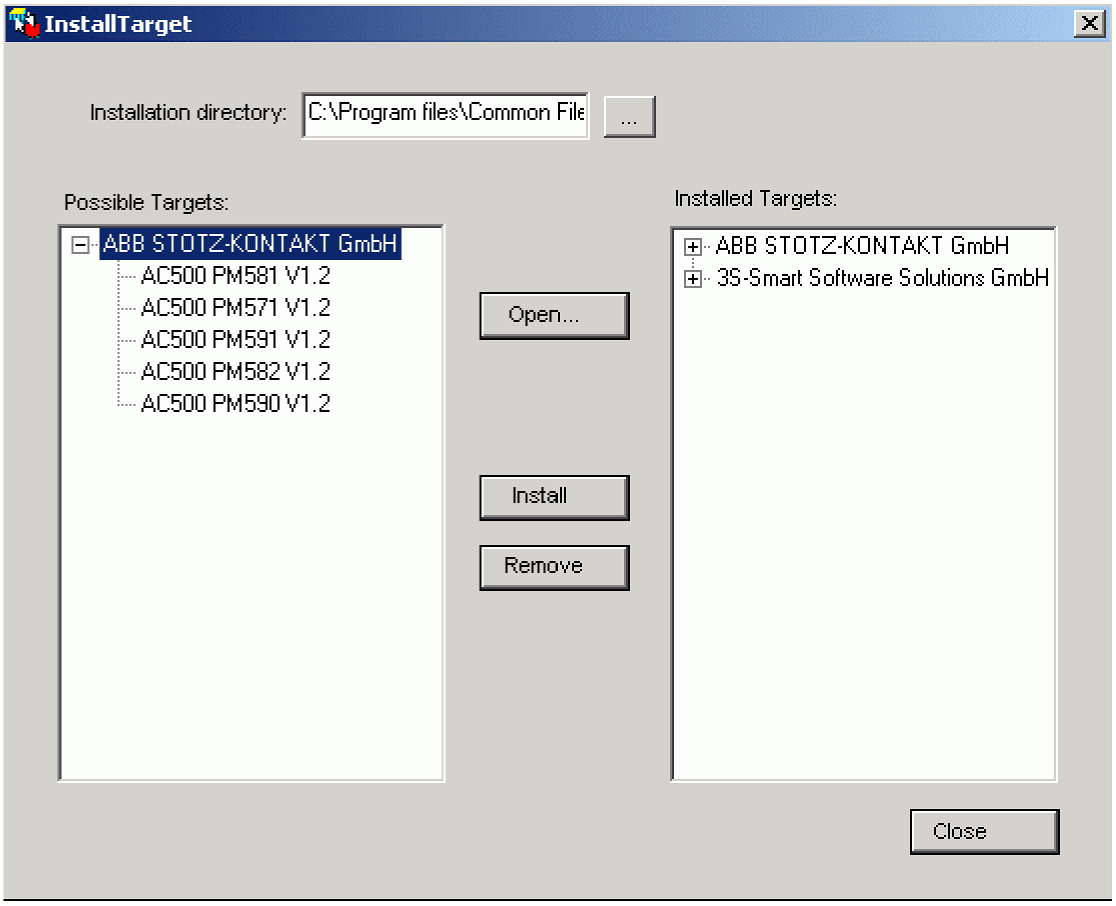
File name: AC500_V12.tnf

Files of type: Target Information File (*.TNF)

The following target installation files are available for the AC500 targets:
AC500_multi.tnf (PM571, PM581 and PM591 V1.1)
AC500_PM582.tnf (PM581 V1.1)
AC500_V12.tnf (PM571, PM581, PM582, PM590, PM591 V1.2)

Click <Open>.

7. In the left-hand area, all targets that are part of this installation file are displayed:



Select "ABB STOTZ-KONTAKT GmbH" or a specific target (for example AC500 PM581 V1.2) and click <Install>. All files belonging to the target will be installed.
The "Installation directory" field shows the target directory.

8. To install further targets repeat steps 5 to 7.

9. Click <Close> to exit the program.

10. If you want to uninstall one or several targets, select the desired target(s) in the "Installed targets" area and click <Remove>.

    Remark:
    "Remove" does only delete the Windows registry entries. The files are still available in the installation directory. They have to be deleted manually, if necessary.

2 AC500 inputs, outputs and flags

All operands supported by CoDeSys are described in the Control Builder documentation. The documentation you are reading here describes in detail the "address" operands (%I for inputs and %Q for outputs) used in CoDeSys.

All addressable operands can be accessed bit-wise (X), byte-wise (B), word-wise (W) and double-word-wise (D) in the Control Builder. The Motorola byteorder is used for operand access.

Declaration of addressable operands:

The declaration of the operands in the addressable flag area is done as follows:

Symbol AT address : Type [:= initialization value]; (* comment *)

[.] optional

The inputs and outputs are declared using the PLC configuration. Input and output devices that are directly coupled to the base unit are declared directly in the PLC configuration. Input and output devices connected to the coupler are configured using the field bus configurator SYCON.net which is part of the Control Builder (see topic Controller configuration with the Control Builder).

⚠ **Caution:**

For multitasking, the digital inputs and outputs for every task are byte-wise cycle consistent, i.e., for instance inputs %IX0.0-%IX0.7 for task 1 and %IX1.0-%IX1.7 for task 2.

If, for example, task 1 has the higher priority and input %IX0.0 is used in task 1 and task 2, the value can change during the cycle of task 2 as it is updated every time task 1 is started.

This is not relevant for programs with only one task.

2.1 AC500 interfaces for inputs and outputs AC500

The following AC500 interfaces are available for inputs and outputs:

No.	Type	Designation	Number of inputs and outputs
Configuration with CoDeSys PLC configuration (ConfConf)			
1	I/O bus	Interface for I/O modules	Max 7 modules with a maximum of 32 channels (IX, QX, IW, QW) per module
2	COM1	CS31 bus master	Max 31 modules with a maximum of 32 channels per module, address 0-61
		Decentralized I/O expansion	RS-232 / RS-485 (version V2.0 and later)
3	COM2	Decentralized I/O expansion	RS-232 / RS-485 (version V2.0 and later)
4	FBP	FieldBusPlug - Slave	Max 8 modules with 16 IW + 16 QW + 16 IB + 16 QB with modular FBP, depending on fieldbus
5	Int. coupler	Internal coupler	ARCNET,.. (configuration without SYCON.net)
Configuration with integrated SYCON.net			
6	Line 0	Internal coupler	4 kB %I0.xx / %Q0.xx each
7	Line 1	Coupler 1	4 kB %I1.xx / %Q1.xx each
8	Line 2	Coupler 2	4 kB %I2.xx / %Q2.xx each
9	Line 3	Coupler 3	4 kB %I3.xx / %Q3.xx each
10	Line 4	Coupler 4	4 kB %I4.xx / %Q4.xx each

2.1.1 Address scheme for inputs and outputs

- The coupler I/Os are addressed as follows (two-stage process):

  %I(Q)B**CouplerNumber.ByteCoupler**

  The configuration is done using SYCON.net.

- No coupler numbers are assigned to I/Os that are connected to the CPU. These I/Os are configured with the PLC configuration (ConfConf) in the Control Builder.

- I/Os connected to the basic unit are assigned to the following address areas:

I/O bus:	%IB0 ..	%IB999	and %QB0 ..	%QB999
COM1:	%IB1000 ..	%IB1999	and %QB1000 ..	%QB1999
COM2 :	%IB2000 ..	%IB2999	and %QB2000 ..	%QB2999
FBP slave:	%IB3000 ..	%IB3999	and %QB3000 ..	%QB3999

- Addressing of the digital channels is done byte-oriented.

- Motorola byteorder is used to access the inputs and outputs.

2.1.2 Example for addressing in BOOL / BYTE / WORD / DWORD

The Motorola byteorder is used for addressing.

Address	**Addr**	**Addr + 1**	**Addr +2**	**Addr +3**
	16#xxxx x000	16#xxxx x001	16#xxxx x002	16#xxxx x003
BYTE	**%IB0**	**%IB1**	**%IB2**	**%IB3**
BOOL	7 ... 0	7 ... 0	7 ... 0	7 ... 0
	%IX0.7 ... %IX0.0	%IX1.7 ... %IX1.0	%IX2.7 ... %IX2.0	%IX3.7 ... %IX3.0
WORD	**%IW0**		**%IW1**	
	15 ... 8	7 ... 0	15 ... 8	7 ... 0
DWORD	**%ID0**			
	31 ... 24	23 ... 16	15 ... 8	7 ... 0

Examples:

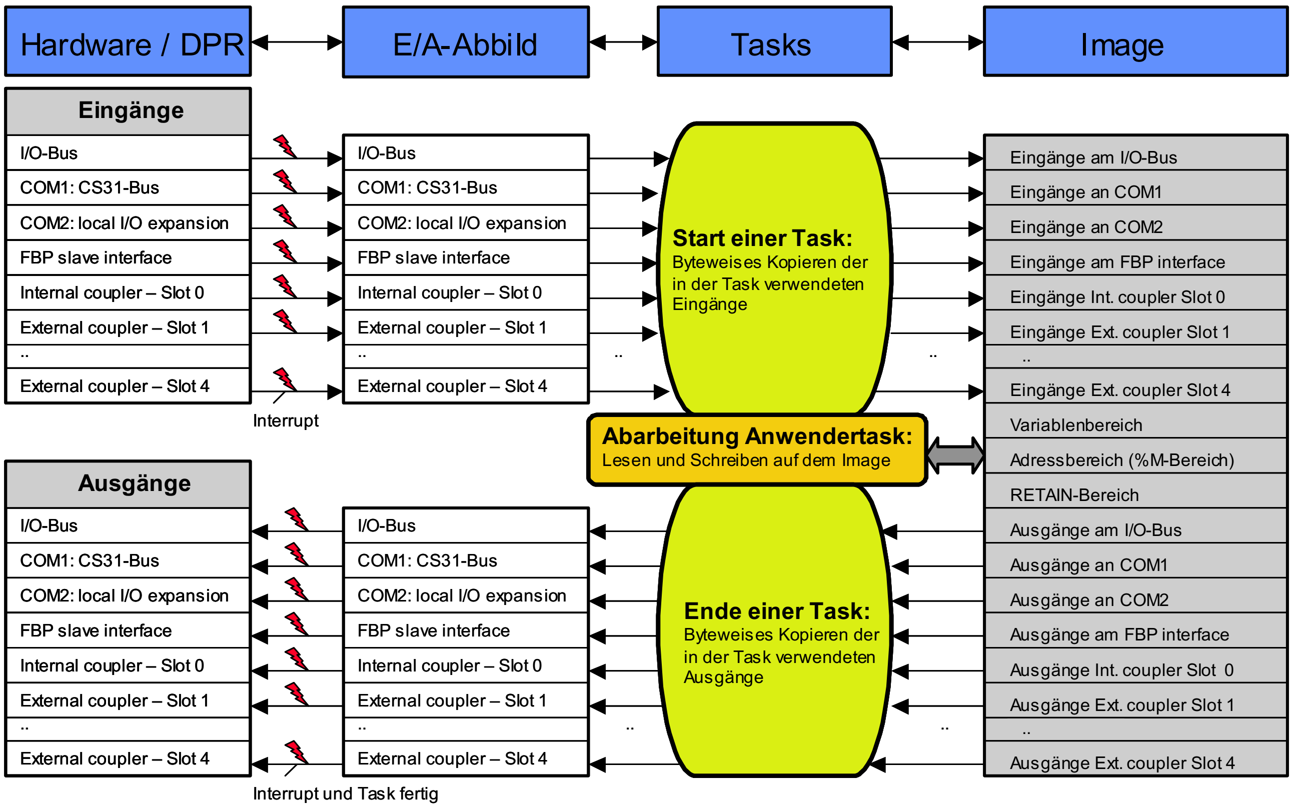
%IX0.0	:= TRUE		
	%IB0 := 1	:= 16#01	
	%IW0 := 256	:= 16#0100	(Bit 8 = TRUE)
	%ID0 := 16777216	:= 16#01000000	(Bit 24 = TRUE)
%IX3.0	:= TRUE		
	%IB3 := 1	:= 16#01	
	%IW1 := 1	:= 16#0001	
	%ID0 := 1	:= 16#00000001	

2.2 Addressing of inputs and outputs

No.	Device	Input/Output	Interface		Range	Addresses
Configuration with ConfConf (CPU I/Os) or SYCON.net (internal coupler)						
0 ... 5	CPU I/Os and	Inputs (4kB)	C P U	I/O bus COM1 COM2 FBP	0000..0999 1000..1999 2000..2999 3000..4095	%IB0 ... %IB4095 %IW0 ... %IW2047 %ID0 ... %ID1023 %IX0.0 ... %IX4095.7
		Outputs (4kB)		I/O bus COM1 COM2 FBP	0000..0999 1000..1999 2000..2999 3000..4095	%QB0 ... %QB4095 %QW0 ... %QW2047 %QD0 ... %QD1023 %QX0.0 ... %QX4095.7
	Internal coupler	Inputs (4kB)	Line 0		0.0000 ... 0.4095	%IB0 ... %IB4095 %IW0 ... %IW2047 %ID0 ... %ID1023 %IX0.0 ... %IX4095.7
		Outputs (4kB)				%QB0 ... %QB4095 %QW0 ... %QW2047 %QD0 ... %QD1023 %QX0.0 ... %QX4095.7
Configuration with SYCON.net						
6	Coupler 1	Inputs (4kB)	Line 1		1.0000 ... 1.4095	%IB1.0 ... %IB1.4095 %IW1.0 ... %IW1.2047 %ID1.0 ... %ID1.1023 %IX1.0.0 ... %IX1.4095.7
		Outputs (4kB)				%QB1.0 ... %QB1.4095 %QW1.0 ... %QW1.2047 %QD1.0 ... %QD1.1023 %QX1.0.0 ... %QX1.4095.7
...						
9	Coupler 4	Inputs (4kB)	Line 4		4.0000 ... 4.4095	%IB4.0 ... %IB4.4095 %IW4.0 ... %IW4.2047 %ID4.0 ... %ID4.1023 %IX4.0.0 ... %IX4.4095.7
		Outputs (4kB)				%QB4.0 ... %QB4.4095 %QW4.0 ... %QW4.2047 %QD4.0 ... %QD4.1023 %QX4.0.0 ... %QX4.4095.7

2.3 Processing of inputs and outputs in the multitasking system

The following figure shows how the inputs and outputs are processed in the multitasking system.



Hardware / DPR
E/A-Abbild
Tasks
Image
Eingänge
I/O-Bus
COM1: CS31-Bus
COM2: local I/O expansion
FBP slave interface
Internal coupler – Slot 0
External coupler – Slot 1
External coupler – Slot 4
Interrupt
Start einer Task:
Byteweises Kopieren der in der Task verwendeten Eingänge
Eingänge am I/O-Bus
Eingänge an COM1
Eingänge an COM2
Eingänge am FBP interface
Eingänge Int. coupler Slot 0
Eingänge Ext. coupler Slot 1
Eingänge Ext. coupler Slot 4
Abarbeitung Anwendertask:
Lesen und Schreiben auf dem Image
Variablenbereich
Adressbereich (%M-Bereich)
RETAIN-Bereich
Ausgänge
Ende einer Task:
Byteweises Kopieren der in der Task verwendeten Ausgänge
Ausgänge am I/O-Bus
Ausgänge an COM1
Ausgänge an COM2
Ausgänge am FBP interface
Ausgänge Int. coupler Slot 0
Ausgänge Ext. coupler Slot 1
Ausgänge Ext. coupler Slot 4
Interrupt und Task fertig

Generation of the input data image:

Inputs at the I/O bus:

After all I/O modules have been processed at the I/O bus, a corresponding interrupt is generated in the processor. The inputs are copied to the input data image during the Interrupt Service Routine (ISR). If the outputs were updated by a task, the outputs in the output data image are copied.

Inputs at the CS31 system bus:

After the CS31 driver has processed all I/O modules, a corresponding interrupt is generated in the processor. The inputs are copied to the input data image during the Interrupt Service Routine (ISR). If the outputs were updated by a task, the outputs in the output data image are copied.

Inputs of couplers line 0 to 4:

Once a coupler has received new data, a corresponding interrupt is generated in the processor. The inputs are copied from the DPR to the input data image of the processor during the Interrupt Service Routine (ISR). If the outputs were updated by a task, the outputs in the output data image are copied to the DPR.

Precondition for this is a valid coupler configuration.

Starting a task:

When starting a task, **the inputs used in the task are copied byte-wise** from the input data image to the image. Byte-wise means that when using, for example, the input %IX0.0, the image of the inputs %IX0.0 ... IX0.7 will be copied to the image.

Because only those inputs are copied that are directly used in the task, it is not possible to **read the inputs indirectly**, if cycle consistency is required.

Processing a task:

All tasks access the image, i.e., inputs are read from the image and outputs are written to the image. In ONLINE mode, the inputs/outputs of the image are displayed.

Termination of processing the output data image by a task:

At the end of the task processing, the outputs used in the task are copied byte-wise from the image to the output data image. Byte-wise means that when using, for example, the output %QX0.0, the image of the outputs %QX0.0 ... QX0.7 will be copied from the image to the output data image. The internal variables "Output data image updated" for the CS31 processor and the couplers 0 .. 4 will be set.

Writing the outputs:

Outputs at the I/O bus:

With the next interrupt of the I/O bus driver, the outputs of the output data image will be written and the variable "Output data image updated" will be reset.

Outputs at the CS31 system bus:

With the next interrupt of the CS31 processor, the outputs of the output data image will be written and the variable "Output data image updated" will be reset.

Outputs of the coupler line 0 to 4:

With the next interrupt of the coupler, the outputs of the output data image will be written to the DPR and the variable "Output data image updated" will be reset.

I/O update task:

In order to update the inputs/outputs not used in the task, all inputs/outputs of the image are updated by a lower priority task (I/O update task). This task is only processed if no other user task runs.

2.4 Addressable flag area (%M area) in the AC500

2.4.1 Allocation of the addressable flag area in the AC500

The addressable flag area for the AC500 is divided into several segments with a size of 64 kbytes per segment. A maximum of 8 segments can be addressed. The availability of the segments or partial segments depends on the CPU. The size of the %M area can be found in the technical data of the CPUs (see Technical data of the CPU) and in the target system settings (see Target Support Package).

Segment	Operands	Size, cumulative [kB]	CPU PM57x	CPU PM58x	CPU PM59x
0	%MB0.0...%MB0.65535	64	4 kB	+	+
1	%MB1.0...%MB1.65535	128	-	+	+
2	%MB2.0...%MB2.65535	192	-	-	+
3	%MB3.0...%MB3.65535	256	-	-	+
4	%MB4.0...%MB4.65535	320	-	-	+
5	%MB5.0...%MB5.65535	284	-	-	+
6	%MB6.0...%MB6.65535	448	-	-	+
7	%MB7.0...%MB7.65535	512	-	-	+

2.4.2 Access to the %M area using the Modbus® Protocol

The Modbus® RTU protocol is implemented in the AC500. With the help of the Modbus® protocol, the segments 0 and 1 of the addressable flag area can be accessed.

The chapter Modbus in this documentation contains a detailed description of the Modbus® protocol and the corresponding addressing (see also Modbus protocol).

Note: For the AC500 CPU PM571, 4kB = %MB0.0 .. %MB0.4095 (i.e., not a complete segment) are available for the addressable flag area. Thus, not all Modbus addresses can be accessed.

2.4.3 Access to operands in the addressable flag area

The operands in the %M area can be accessed bit-wise, byte-wise, word-wise and double-word-wise.

Byte **SINT / BYTE**	**Bit (byte-oriented)** **BOOL**	**Word** **INT / WORD**	**Double word** **DINT / DWORD**
Segment 0			
%MB0.0	%MX0.0.0 ... %MX0.0.7	%MW0.0	%MD0.0
%MB0.1	%MX0.1.0 ... %MX0.1.7		
%MB0.2	%MX0.2.0 ... %MX0.2.7	%MW0.1	
%MB0.3	%MX0.3.0 ... %MX0.3.7		
...	...	...	...
%MB0.65532	%MX0.65532.0 ... %MX0.65532.7	%MW0.32766	%MD0.16383
%MB0.65533	%MX0.65533.0 ... %MX0.65533.7		
%MB0.65534	%MX0.65534.0 ... %MX0.65534.7	%MW0.32767	
%MB0.65535	%MX0.65535.0 ... %MX0.65535.7		
Segment 1			
%MB1.0	%MX1.0.0 ... %MX1.0.7	%MW1.0	%MD1.0
%MB1.1	%MX1.1.0 ... %MX1.1.7		
%MB1.2	%MX1.2.0 ... %MX1.2.7	%MW1.1	
%MB1.3	%MX1.3.0 ... %MX1.3.7		
...	...	...	...
%MB1.65532	%MX1.65532.0 ... %MX1.65532.7	%MW1.32766	%MD1.16383
%MB1.65533	%MX1.65533.0 ... %MX1.65533.7		
%MB1.65534	%MX1.65534.0 ... %MX1.65534.7	%MW1.32767	
%MB1.65535	%MX1.65535.0 ... %MX1.65535.7		
Segment 2			
%MB2.0	%MX2.0.0 ... %MX2.0.7	%MW2.0	%MD2.0
%MB2.1	%MX2.1.0 ... %MX2.1.7		
%MB2.2	%MX2.2.0 ... %MX2.2.7	%MW2.1	
%MB2.3	%MX2.3.0 ... %MX2.3.7		
...	...	...	...
%MB2.65532	%MX2.65532.0 ... %MX2.65532.7	%MW2.32766	%MD2.16383
%MB2.65533	%MX2.65533.0 ... %MX2.65533.7		
%MB2.65534	%MX2.65534.0 ... %MX2.65534.7	%MW2.32767	
%MB2.65535	%MX2.65535.0 ... %MX2.65535.7		
...	...	...	...
Segment 7			
%MB7.0	%MX7.0.0 ... %MX7.0.7	%MW7.0	%MD7.0
%MB7.1	%MX7.1.0 ... %MX7.1.7		
%MB7.2	%MX7.2.0 ... %MX7.2.7	%MW7.1	
%MB7.3	%MX7.3.0 ... %MX7.3.7		
...	...	...	...
%MB7.65532	%MX7.65532.0 ... %MX7.65532.7	%MW7.32766	%MD7.16383
%MB7.65533	%MX7.65533.0 ... %MX7.65533.7		
%MB7.65534	%MX7.65534.0 ... %MX7.65534.7	%MW7.32767	
%MB7.65535	%MX7.65535.0 ... %MX7.65535.7		

2.5 Absolute addresses of operands

2.5.1 Adress operator ADR

For particular blocks or in case of accessing operands via pointers, the absolute address of an operand must be determined. To do this, the Control Builder provides the address operator ADR.

The address operator ADR is described in the documentation for the Control Builder (see CoDeSys Documentation / ADR operator). The documentation you are reading here describes only the peculiarities of bit operands.

The addresses provided by the address operator can be used as inputs for blocks that require absolute addresses (such as xxx_MOD_MAST, COM_SND). If these blocks shall be applied to internal variables, it must be guaranteed that the variables are set to successive addresses. This is achieved by declaring ARRAYs and STRINGs.

The address operator ADR provides the address of an operand in one double word DWORD (i.e., 32 bits). The address operator returns the address of the first byte of a variable (byte address). For the user-definable variables, variables of the type BOOL are stored as byte.

2.5.2 Bit address operator BITADR

For inputs, outputs and variables of the addressable flag area (%M area) or addressable PERSISTENT area (%R area), operands of the type BOOL occupy one bit. The address of this type of variables cannot be determined with the operator ADR.

When processing the statement:

```
dwAddress := ADR(%MX0.0.0);
```

the following error message appears:

```
Error 4031:
PLC_PRG(xx): ADR is not allowed for bits! Use BITADR instead.
```

BITADR returns the bit offset within the area %I, %Q or %M as DWORD.

The following table shows the position of the operands within the memory (considering %MD0.0 and %MD0.1 as example). Here you get information about which addresses the operator ADR returns and which offsets BITADR returns.

Note: The addresses shown are example addresses and thus can have other values.

Position of operands within memory and values of operators ADR and BITADR:

Byte SINT / BYTE	Word INT / WORD	Double word DINT / DWORD	Bit (byte-oriented) BOOL	ADR	BITADR
%MB0.0	%MW0.0	%MD0.0	%MX0.0.0	16#08000000	8
			%MX0.0.1		9
			%MX0.0.2		10
			%MX0.0.3		11
			%MX0.0.4		12
			%MX0.0.5		13
			%MX0.0.6		14
			%MX0.0.7		15
%MB0.1			%MX0.1.0	16#08000001	0
			%MX0.1.1		1
			%MX0.1.2		2
			%MX0.1.3		3
			%MX0.1.4		4
			%MX0.1.5		5
			%MX0.1.6		6
			%MX0.1.7		7
%MB0.2	%MW0.1		%MX0.2.0	16#08000002	24
			%MX0.2.1		25
			%MX0.2.2		26
			%MX0.2.3		27
			%MX0.2.4		28
			%MX0.2.5		29
			%MX0.2.6		30
			%MX0.2.7		31
%MB0.3			%MX0.3.0	16#08000003	16
			%MX0.3.1		17
			%MX0.3.2		18
			%MX0.3.3		19
			%MX0.3.4		20
			%MX0.3.5		21
			%MX0.3.6		22
			%MX0.3.7		23
%MB0.4	%MW0.2	%MD0.1	%MX0.4.0	16#08000004	40
			..		..
			%MX0.4.7		47
%MB0.5			%MX0.5.0	16#08000005	32
			..		..
			%MX0.5.7		39
%MB0.6	%MW0.3		%MX0.6.0	16#08000006	56
			..		..
			%MX0.6.7		63
%MB0.7			%MX0.7.0	16#08000007	48
			..		..
			%MX0.7.7		55

2.6 Addressable PERSISTENT area (%R area) in the AC500

2.6.1 Special features of the addressable PERSISTENT area in the AC500

As of Control Builder version V1.2 and CPU firmware V1.2.0, the new operand area "addressable PERSISTENT area" or %R area is available.

> **Caution:** The %R area is only available in combination with Control Builder version V1.2 and later and AC500 firmware version V1.2.0 and later. For the %R area the following is relevant for Control Builder V1.2:
> **CoDeSys.exe V2.3.8.0** or higher (shown under CoDeSys / Info)
> **AC500 Target PM5xx_V12** or higher (shown under InstallTarget / Installed targets)

The addressable PERSISTENT area or %R area has the following **peculiarities**:

1. Variables declared in the %R area are always located at the same position in the PLC's operand memory because they have addresses assigned (like the variables in the %R area).

2. Variables in the %R area are declared as follows:

   VAR (**Caution: noRETAIN** or **PERSISTENT option)**
   Symbol AT %RTypeSegment.Offset : TYPE; (* Comment *) or also
   aSymbol AT %RTypeSegment.Offset : ARRAY[start..end] OF TYPE; (* Comment *)
   END_VAR

where:	Symbol	- symbolic name of the variable
	Type	- X=BOOL (Bit), B=BYTE, W=WORD, D=DWORD
	Segment	- 0..7 (availability depends on CPU type)
	Offset	- 0..65535 (availability depends on CPU type)
	TYPE	- BOOL, BYTE, WORD, DWORD or defined type (such as structure)
	start	- Index of the first ARRAY element
	end	- Index of the last ARRAY element

3. For each segment in the %R area, an area can be set in the PLC configuration which is buffered in case the **battery is installed and fully charged**. In this case, the variables behave like variables declared as VAR RETAIN PERSISTENT, i.e.,
   -> they keep their values even after
   - Online changes (like VAR RETAIN)
   - Voltage OFF/ON (like VAR RETAIN)
   - a download (like VAR PERSISTENT)

4. In contrast to the variables declared as PERSISTENT, these variables have the great advantage that no program code is required for dumping the variables during a download.

5. The buffered part of the %R area can be written to the SD card and read from the card (see chapter "Saving the buffered data of the %R area").

2.6.2 Segmentation of the addressable PERSISTENT area in the AC500

The addressable PERSISTENT area in the AC500 is divided into several segments with a size of 64 kbytes per segment. A maximum of 8 segments can be addressed. The availability of the segments or partial segments depends on the CPU:

Segment	Operands	Size, cumulative [kB]	CPU PM57x	CPU PM58x	CPU PM59x
0	%RB0.0...%RB0.65535	64	4 kB	+	+
1	%RB1.0...%RB1.65535	128	-	+	+
2	%RB2.0...%RB2.65535	192	-	-	+
3	%RB3.0...%RB3.65535	256	-	-	+
4	%RB4.0...%RB4.65535	320	-	-	+
5	%RB5.0...%RB5.65535	284	-	-	+
6	%RB6.0...%RB6.65535	448	-	-	+
7	%RB7.0...%RB7.65535	512	-	-	+

2.6.3 Saving the buffered data of the AC500's %R area

The buffered part of the %R area can be saved on the SD card and read from the card. This can be necessary, if, for example, the controller has to be replaced.

Saving data is done in two steps:

1. Copying the data from the %R area and writing it to the CPU's RAM disk as file
2. Saving the file to the SD card.

Reading data from the SD card is also done in two steps:

1. Loading the file from the SD card to the CPU's RAM disk.
2. Copying the data from the RAM disk to the %R area.

Saving and reading the data can be done using function blocks in the user program or with the PLC Browser contained in the Control Builder. The function blocks are contained in the library SysInt_AC500_V10.LIB.

Function	PLC Browser command	Function block
Copy from %R area to RAM disk	persistent save	PERSISTENT_SAVE
Save file to SD card	persistent export	PERSISTENT_EXPORT
Read file from SD card to RAM disk	persistent import	PERSISTENT_IMPORT
Copy data from RAM disk to %R area	persistent restore	PERSISTENT_RESTORE
Delete buffered data of the PERSISTENT area	persistent clear	PERSISTENT_CLEAR

⚠ **Caution:** If cycle consistency is required for the data, this has to be implemented in the user program. That means that the data may not be changed during copying to/from the %R area from/to the RAM disk.
If saving is done using the PLC Browser, this can be easily carried out by stopping the user program.

⚠ **Caution:** Copying the PERSISTENT area takes some milliseconds (see the following table). Thus, an according cycle time has to be set in the task configuration. Please note the remarks on the task configuration!

Action	**Time in ms**		
	CPU PM57x	**CPU PM58x**	**CPU PM59x**
Restoring 1 kB (1024 bytes)			
PERSISTENT_CLEAR	< 1	< 1	< 1
PERSISTENT_SAVE	2	2	2
PERSISTENT_EXPORT	1000	1000	500
PERSISTENT_IMPORT	500	1000	500
PERSISTENT_RESTORE	2	< 1	1
Restoring 4 kB (4096 bytes)			
PERSISTENT_CLEAR	< 1	< 1	< 1
PERSISTENT_SAVE	2	3	2
PERSISTENT_EXPORT	1000	1000	500
PERSISTENT_IMPORT	500	1000	500
PERSISTENT_RESTORE	3	3	2
Restoring 64 kB (65536 bytes)			
PERSISTENT_CLEAR	not possible	8	2
PERSISTENT_SAVE	not possible	11	6
PERSISTENT_EXPORT	not possible	2500	1000
PERSISTENT_IMPORT	not possible	2000	500
PERSISTENT_RESTORE	not possible	12	5
Restoring max. PERSISTENT area			
	4 kB	**128 kB**	**512 kB**
PERSISTENT_CLEAR	< 1	17	22
PERSISTENT_SAVE	2	22	35
PERSISTENT_EXPORT	1000	4000	8000
PERSISTENT_IMPORT	500	3000	4000
PERSISTENT_RESTORE	3	22	31

2.6.4 Access to operands in the addressable PERSISTENT area (%R area)

The operands in the %R area can be accessed bit-wise, byte-wise, word-wise and double-word-wise.

Byte SINT / BYTE	Bit (byte-oriented) BOOL	Word INT / WORD	Double word DINT / DWORD
Segment 0			
%RB0.0	%RX0.0.0...%RX0.0.7	%RW0.0	%RD0.0
%RB0.1	%RX0.1.0...%RX0.1.7		
%RB0.2	%RX0.2.0...%RX0.2.7	%RW0.1	
%RB0.3	%RX0.3.0...%RX0.3.7		
%RB0.65532	%RX0.65532.0...%RX0.65532.7	%RW0.32766	%RD0.16383
%RB0.65533	%RX0.65533.0...%RX0.65533.7		
%RB0.65534	%RX0.65534.0...%RX0.65534.7	%RW0.32767	
%RB0.65535	%RX0.65535.0...%RX0.65535.7		
Segment 1			
%RB1.0	%RX1.0.0...%RX1.0.7	%RW1.0	%RD1.0
%RB1.1	%RX1.1.0...%RX1.1.7		
%RB1.2	%RX1.2.0...%RX1.2.7	%RW1.1	
%RB1.3	%RX1.3.0...%RX1.3.7		
%RB1.65532	%RX1.65532.0...%RX1.65532.7	%RW1.32766	%RD1.16383
%RB1.65533	%RX1.65533.0...%RX1.65533.7		
%RB1.65534	%RX1.65534.0...%RX1.65534.7	%RW1.32767	
%RB1.65535	%RX1.65535.0...%RX1.65535.7		
Segment 2			
%RB2.0	%RX2.0.0...%RX2.0.7	%RW2.0	%RD2.0
%RB2.1	%RX2.1.0...%RX2.1.7		
%RB2.2	%RX2.2.0...%RX2.2.7	%RW2.1	
%RB2.3	%RX2.3.0...%RX2.3.7		
%RB2.65532	%RX2.65532.0...%RX2.65532.7	%RW2.32766	%RD2.16383
%RB2.65533	%RX2.65533.0...%RX2.65533.7		
%RB2.65534	%RX2.65534.0...%RX2.65534.7	%RW2.32767	
%RB2.65535	%RX2.65535.0...%RX2.65535.7		
Segment 7			
%RB7.0	%RX7.0.0...%RX7.0.7	%RW7.0	%RD7.0
%RB7.1	%RX7.1.0...%RX7.1.7		
%RB7.2	%RX7.2.0...%RX7.2.7	%RW7.1	
%RB7.3	%RX7.3.0...%RX7.3.7		
%RB7.65532	%RX7.65532.0...%RX7.65532.7	%RW7.32766	%RD7.16383
%RB7.65533	%RX7.65533.0...%RX7.65533.7		
%RB7.65534	%RX7.65534.0...%RX7.65534.7	%RW7.32767	
%RB7.65535	%RX7.65535.0...%RX7.65535.7		

Note: Only the first 4 kB in segment 0 are available for PM57x, i.e., %RB0.0..%RB0.4095 or %RW0.0..%RW0.2047 or %RD0.0..%RD0.1023.

3 The AC500 PLC configuration

3.1 Overview on the PLC configuration

3.1.1 PLC configuration functions

The general operation of the PLC configuration is described in detail in the Control Builder documentation (see CoDeSys Documentation / PLC Configuration). This section describes the configuration of the AC500.

The PLC configuration describes the hardware of the project. This way, the following data can be made available in the project:

- General parameters of the AC500 CPU
- Inputs and outputs of all modules connected to the I/O bus
- Symbolic names and comments for the inputs and outputs
- Parameters of the input and output modules and the assigned I/O channels, if available
- Mode and parameter settings for the serial interfaces
- Inputs and outputs of all input/output modules connected to the serial interface COM1 in CS31 mode
- Coupler type, general parameters and logs of the installed couplers
- All required system libraries are automatically loaded according to the configuration when building the project (started by pressing <F11>)
- Creation of a new database for exporting and importing configuration data in XML data format

The PLC configuration for the AC500 allows to load a project into all AC500 CPUs (PM571, PM581, PM591, ..). In order to download a project to another CPU, the desired CPU has to be selected in the target settings. It is only necessary to change the PLC configuration, if the hardware structure of the PLC has changed, i.e., if, for example, other couplers are installed.

The PLC configuration allows to export and import the complete configuration or parts of it. Thus, it is for example possible to store previously edited symbolic names of the inputs/outputs of an I/O module on the PC and to import them into other projects. This is also possible for the interface settings (see also chapter "Export and import of configuration data").

3.1.2 Export and import of configuration data

Exporting a module, all interface data or a complete configuration is done by selecting the desired element in the PLC configuration, opening the context menu by right-clicking the element and selecting the menu item "Export module". In the appearing window, the module can be saved to the corresponding configuration level as XML file under the selected module name.

Importing modules is done in the same way than exporting. That means, by right-clicking the desired module in the configuration tree and selecting the menu item "Import module" from the context menu.

Note: Modules can only be imported into the configuration level from which they were previously exported.

In case of I/O modules (such as I/O bus, CS31 bus), the position of the module can be ignored. Thus, a module installed in slot 1 can be exported and imported to slot 5. The input and output addresses are changed automatically.

Example of an export/import procedure:

Example 1:

The input/output module DC532 at the I/O bus for machine part Axx shall be saved as DC532_Axx and then be used in another project.

1. Append a DC532 module to the I/O bus by right-clicking the I/O bus element in the configuration tree and selecting the menu item "Append Subelement" / "DC532 - 16 digital input and 16 digital Inoutput" from the context menu.
2. Edit the symbolic names and comments of the inputs/outputs.
3. Export the DC532 module by right-clicking it in the configuration tree and selecting the menu item "Export module" from the context menu. In the appearing window, enter the file name "DC532_Axx" and confirm the window.
   Now the file "DC532_Axx.xml" is saved to the directory Compile which is a subdirectory of the Control Builder installation directory.
4. If necessary, save the project and open/create the project into which you want to insert the exported module.
5. Append a DC532 module to the I/O bus of this project.
6. Right-click the module DC532 in the configuration tree, select the context menu item "Import module" and then choose the file "DC532_Axx" (contains the exported module) in the appearing window. The module will be inserted and the symbolic names and comments are available in the projects afterwards.

Example 2:

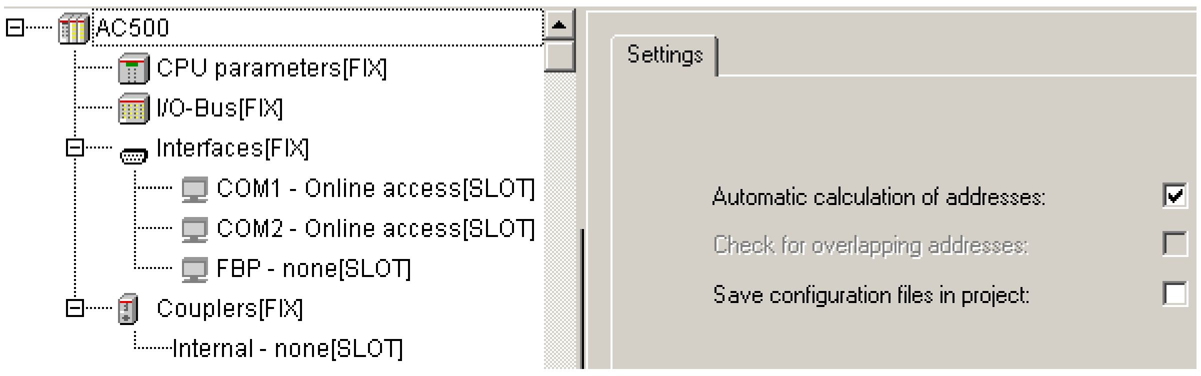
The Modbus RTU settings for serial interface COM1 shall be saved as "COM1_MODBUS_Slave1".

1. Change the COM1 parameters to "COM1 - MODBUS" by right-clicking the element in the configuration tree and selecting "Replace element" / "COM1 - MODBUS".
2. Configure the interface parameters in the "Module parameters" tab.
3. Export the new settings by right-clicking COM1 in the configuration tree and selecting the menu item "Export module" from the context menu. In the appearing window, enter the file name "COM1_MODBUS_Slave1" and confirm the window.
   Now the file "COM1_MODBUS_Slave1.xml" is saved to the directory Compile of the Control Builder.

3.1.3 Default settings in the PLC configuration

Once you have selected the AC500 CPU in the target settings, the CPU can be configured. To do this, select the object "PLC Configuration" in the "Resources" tab.

In case of a new AC500 project, the PLC configuration contains the following default settings:



```
AC500
    CPU parameters[FIX]
    I/O-Bus[FIX]
    Interfaces[FIX]
        COM1 - Online access[SLOT]
        COM2 - Online access[SLOT]
        FBP - none[SLOT]
    Couplers[FIX]
        Internal - none[SLOT]
```

Settings

Automatic calculation of addresses: [✓]
Check for overlapping addresses: [ ]
Save configuration files in project: [ ]

Note: These default settings can be restored at any time by selecting "Extras" / "Standard configuration". **Do not change the default settings under "Settings"!** As of PS501 version V1.2, the parameter "Automatic calculation for addresses" is no longer editable.

The parameters represent the interfaces of the AC500 controllers. Each interface can be configured.

The individual parameters are used to configure the following elements:

Element	Configuration
CPU parameters	CPU parameters.
I/O bus	Input/output modules that are directly connected to the CPU.
Interfaces	Serial interfaces COM1 and COM2 and FBP slave interface.
Couplers	Parameters and protocols of the internal coupler and the external couplers. The real coupler configuration (ARCNET excluded) and the configuration of the connected input/output modules is done with the integrated fieldbus configurator SYCON.net (see also fieldbus configuration with SYCON.net).

The PLC configuration is based on the configuration files (*.cfg) installed with the TSP.

3.1.4 Setting parameters in the PLC configuration

For all windows containing module parameters the following basic rules apply:

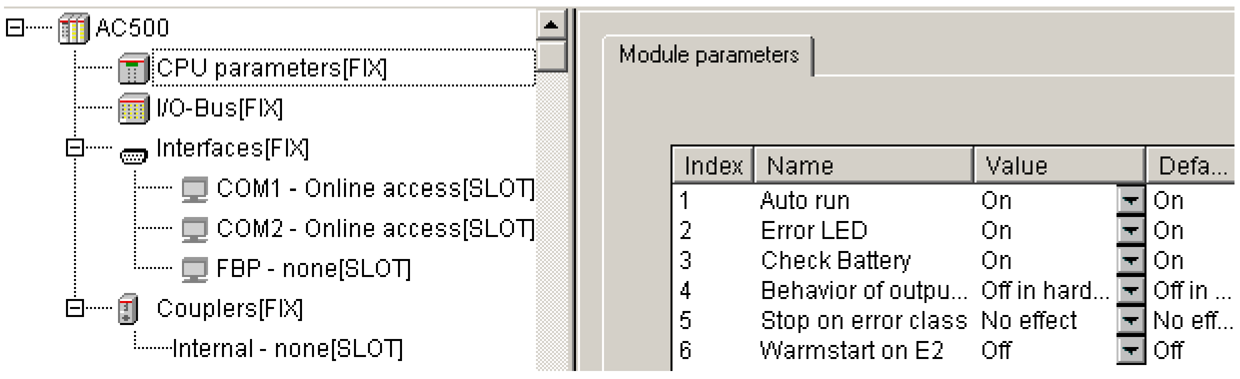
- All visible parameters of the configuration file are displayed.
  Only the values in the column Value can be edited.
- ***Index:*** The Index is a consecutive number (i) for the parameters within a module.
- ***Name:*** Name of the parameter.
- ***Value :*** Value of the parameter, editable.
  The default value is displayed initially. Values can be set directly or by means of symbolic names. If the entries in the configuration file are not set to 'Read Only', they can be edited. To edit a value, click on the edit field or select one of the entries from the scroll list.
- ***Default:*** Default value of the parameter.
- ***Min.:*** Minimum value of the parameter (applies only if no symbolic names are used).
- ***Max.:*** Maximum value of the parameter (applies only if no symbolic names are used).

A ***tooltip*** may give additional information on the currently selected parameter. This information is displayed according to the language setting for the Control Builder.

3.2 Configuration of CPU parameters

3.2.1 CPU parameters in PS501 versions V1.0 and V1.1

Selecting "CPU parameters" in the configuration tree opens the configuration window as shown as follows:



Index	Name	Value	Defa...
1	Auto run	On	On
2	Error LED	On	On
3	Check Battery	On	On
4	Behavior of outpu...	Off in hard...	Off in ...
5	Stop on error class	No effect	No eff...
6	Warmstart on E2	Off	Off

The following parameters can be set:

Parameter	**Default value**	**Value**	**Meaning**
Auto run / Start of user program when voltage ON **see remark 1**	On	On	If the Flash memory contains a valid project, the project will be loaded into the RAM memory and executed when switching on the controller.
		Off	If the Flash memory contains a valid project, this project will be loaded into the RAM memory but not executed when switching on the controller.
Error LED	On	On	The error LED lights up for errors of all classes.
		Off_by_E4	Warnings (E4) are not indicated by the error LED.
		Off_by_E3	Warnings (E4) and light errors (E3) are not indicated by the error LED.
Check Battery	On	On	The availability of the battery and the battery status are checked. If no battery is available or the battery is empty, a warning (E4) is generated and the LED ERR lights up.
		Off	The battery is not checked. No warning (E4) is generated. This also applies if a battery is installed but empty!
Behaviour of outputs in stop	Off in hardware and online	Off in hardware and online	In case of STOP, all outputs at the hardware and in the online display are set to FALSE or 0.
		Off in hardware and actual state online	In case of STOP, all outputs at the hardware are set to FALSE or 0. The online display indicates the status from the last cycle of the user program.
		Actual state in hardware and online	The status of the last cycle of the user program is kept for the outputs at the hardware and in the online display.
Stop on error class	No effect	No effect	In case of an error, the user program is not stopped.
		E1	In case of a fatal error (E1), the user program is stopped.
		E2	In case of a fatal or serious error (E1-E2), the user program is stopped.
		E3	In case of a fatal, serious or light error (E1-E3), the user program is stopped.
		E4	In case of a fatal, serious or light error (E1-E3) or a warning (E4), the user program is stopped.
Warmstart on E2	Off	Off	In case of a fatal error (E2), no warmstart is performed.
		On	In case of a fatal error (E2), a warmstart is performed automatically. (V1.2.0 and higher)

Remark 1: Setting the parameters Auto run and MOD using the display/keypad

Loading and running the user program also depends on the **setting for the parameter MOD using the display/keypad**. The display/keypad setting always has the higher priority.

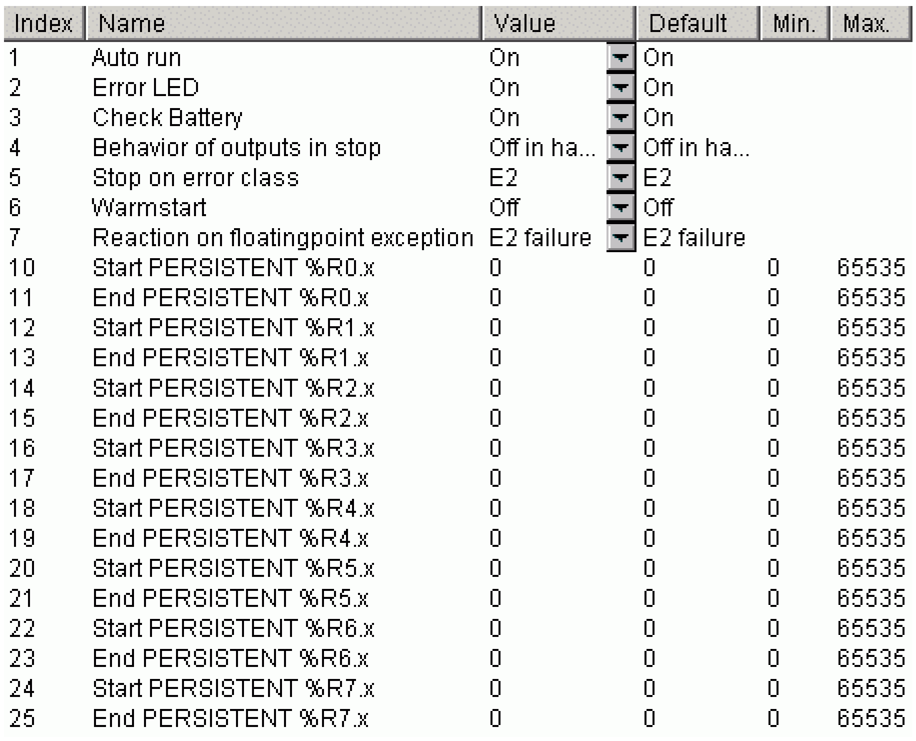
The following applies:

MOD 00:	The user program will be loaded and run according to the setting for the CPU parameter "Auto run" (default setting).
MOD 01:	User program will not be loaded/run.
MOD 02:	The user program will be loaded and run independent of the setting for the CPU parameter "Auto run".

Keeping the "RUN" pushbutton pressed when booting the PLC automatically activates MOD 01, i.e., the user program is not loaded/run. Thus, it is possible to boot the PLC in Stop status. This may be required if, for example, both serial interfaces are set to Modbus and therefore no access with the Control Builder software is possible via the serial interface.

3.2.2 CPU parameters in version PS501 V1.2

The CPU parameters have been revised and expanded for PS501 version V1.2:



Index	Name	Value	Default	Min.	Max.
1	Auto run	On	On		
2	Error LED	On	On		
3	Check Battery	On	On		
4	Behavior of outputs in stop	Off in ha...	Off in ha...		
5	Stop on error class	E2	E2		
6	Warmstart	Off	Off		
7	Reaction on floatingpoint exception	E2 failure	E2 failure		
10	Start PERSISTENT %R0.x	0	0	0	65535
11	End PERSISTENT %R0.x	0	0	0	65535
12	Start PERSISTENT %R1.x	0	0	0	65535
13	End PERSISTENT %R1.x	0	0	0	65535
14	Start PERSISTENT %R2.x	0	0	0	65535
15	End PERSISTENT %R2.x	0	0	0	65535
16	Start PERSISTENT %R3.x	0	0	0	65535
17	End PERSISTENT %R3.x	0	0	0	65535
18	Start PERSISTENT %R4.x	0	0	0	65535
19	End PERSISTENT %R4.x	0	0	0	65535
20	Start PERSISTENT %R5.x	0	0	0	65535
21	End PERSISTENT %R5.x	0	0	0	65535
22	Start PERSISTENT %R6.x	0	0	0	65535
23	End PERSISTENT %R6.x	0	0	0	65535
24	Start PERSISTENT %R7.x	0	0	0	65535
25	End PERSISTENT %R7.x	0	0	0	65535

The following parameters can be set:

Parameter	**Default value**	**Value**	**Meaning**
Auto run / Start of user program when voltage ON **see remark 1**	On	On	If the Flash memory contains a valid project, the project will be loaded into the RAM memory and executed when switching on the controller.
		Off	If the Flash memory contains a valid project, this project will be loaded into the RAM memory but not executed when switching on the controller.
Error LED **see remark 2**	On	On	The error LED lights up for errors of all classes, no failsafe function activated.
		Off_by_E4	Warnings (E4) are not indicated by the error LED, no failsafe function activated.
		Off_by_E3	Warnings (E4) and light errors (E3) are not indicated by the error LED, no failsafe function activated.
		On+failsafe	The error LED lights up for errors of all classes and the failsafe function of the I/O bus is activated.
		Off_by_E4 +failsafe	Warnings (E4) are not indicated by the error LED, the failsafe function of the I/O bus is activated.
		Off_by_E3 +failsafe	Warnings (E4) and light errors (E3) are not indicated by the error LED, the failsafe function of the I/O bus is activated.
Check Battery	On	On	The availability of the battery and the battery status are checked. If no battery is available or the battery is empty, a warning (E4) is generated and the LED ERR lights up.
		Off	The battery is not checked. No warning (E4) is generated. This also applies if a battery is installed but empty!
Behaviour of outputs in stop **see remark 3**	Off in hardware and online	Off in hardware and online	In case of STOP, all outputs at the hardware and in the online display are set to FALSE or 0.
		Off in hardware and actual state online	In case of STOP, all outputs at the hardware are set to FALSE or 0. The online display indicates the status from the last cycle of the user program.
		Actual state in hardware and online	The status of the last cycle of the user program is kept for the outputs at the hardware and in the online display.
Reaction on floating point exceptions **see remark 4**	E2 failure	E2 failure	If a floating point exception occurs, an E2 error (Err=38) is triggered. The CPU goes to STOP. **Warning: PM59x only!**
		No failure	If a floating point exception occurs, no E2 error is triggered. Using the block FPU_EXINFO in the user program allows to react on a possibly occurred exception. **Warning: PM59x only!**

Parameter	Default value	Value	Meaning
Stop on error class **see remark 5**	E2	No effect	In case of a fatal or serious error (E1-E2), the user program is stopped.
		E1	In case of a fatal or serious error (E1-E2), the user program is stopped.
		E2	In case of a fatal or serious error (E1-E2), the user program is stopped.
		E3	In case of a fatal, serious or light error (E1-E3), the user program is stopped.
		E4	In case of a fatal, serious or light error (E1-E3) or a warning (E4), the user program is stopped.
Warmstart **see remark 6**	Off	Off	In case of a fatal error (E2), no warmstart is performed.
		On after E2 error	In case of a fatal error (E2), a warmstart is performed automatically.
		On after short voltage dip	A warmstart is performed after a short voltage dip.
		On after E2 or short voltage dip	In case of a fatal error (E2) or after a short voltage dip, a warmstart is performed automatically.
Start PERSISTENT %R0.x **see remark 7**	0	0..65535	Start offset for buffered area in PERSISTENT area %R0.x
End PERSISTENT %R0.x	0	0...65535	End offset for buffered area in PERSISTENT area %R0.x
...	..	..	..
Start PERSISTENT %R7.x	0	0...65535	Start offset for buffered area in PERSISTENT area %R7.x
End PERSISTENT %R7.x	0	0...65535	End offset for buffered area in PERSISTENT area %R7.x

Remark 1: Setting the parameters Auto run and MOD using the display/keypad

See remark 1 under CPU parameters in PS501 versions V1.0 and V1.1

Remark 2: Error LED

In addition to setting the behavior of the CPU's error LED ERR, this parameter is used to set the failsafe behavior of the I/O bus.

Remark 3: Behaviour of outputs in Stop

The setting of the parameter "Behaviour of outputs in stop" directly influences the failsafe function of the outputs of the S500 I/O devices.

Remark 4: Reaction on floating point exceptions

As of firmware version V1.2.0 of the AC500 CPUs and Control Builder version V1.2, the behavior of the CPUs PM59x regarding floating point exceptions can be set. In standard case, any floating point exception triggers an E2 error: class=E2, err=38, d1=9, d2=31, d3=31.

The CPU goes to STOP.

CPUs without floating point processor PM57x and PM59x do not trigger a floating point exception.

If the parameter "Reaction on floating point exceptions" is set to "No failure", no error is triggered in case of a floating point exception. The CPU remains in RUN mode.

By means of the function block FPU_EXINFO (contained in SysInt_AC500_V10.LIB) it can be determined whether a floating point exception occurred during calculation. Depending on the result, either the calculation can be continued with default values or the machine can be shut down.

Program example:

```
PROGRAM PLC_PRG
VAR
    FPUEXINFO1     : FPU_EXINFO;
    rV1            : REAL := -1.0;
    rV2            : REAL;
    bError         : BOOL;
    bWarning       : BOOL;
END_VAR

bWarning := bError := FALSE;
rV2 := SQRT(rV1);                    (* floating point calculation *)
FPUEXINFO1();                        (* check for exception occurred *)
IF FPUEXINFO1.ERR THEN
    (* evaluation of exception *)
    (* for example, shut down system, continue calculation with default values or corrected values *)
    rV1 := 1.0;
    bWarning := TRUE;
    (* same calculation with corrected values *)
    rV2 := SQRT(rV1);
      FPUEXINFO1();                  (* recheck.. *)
    IF FPUEXINFO1.ERR = TRUE THEN
      bError := TRUE;
    END_IF
END_IF
(* here, for example, evaluation of bWarning, bError.. *)
```

Remark 5: Stop on error class

As of firmware version V1.2.0 of the AC500 CPUs, the user program is stopped with any serious error (class E2) independent of the setting for the parameter "Stop on error class". The settings "No effect" and "E1" have the same behavior as the setting "E2".

The texts could not be changed due to downward compatibility to PS501 V1.0 and V1.1 projects.

Remark 6: Warm start

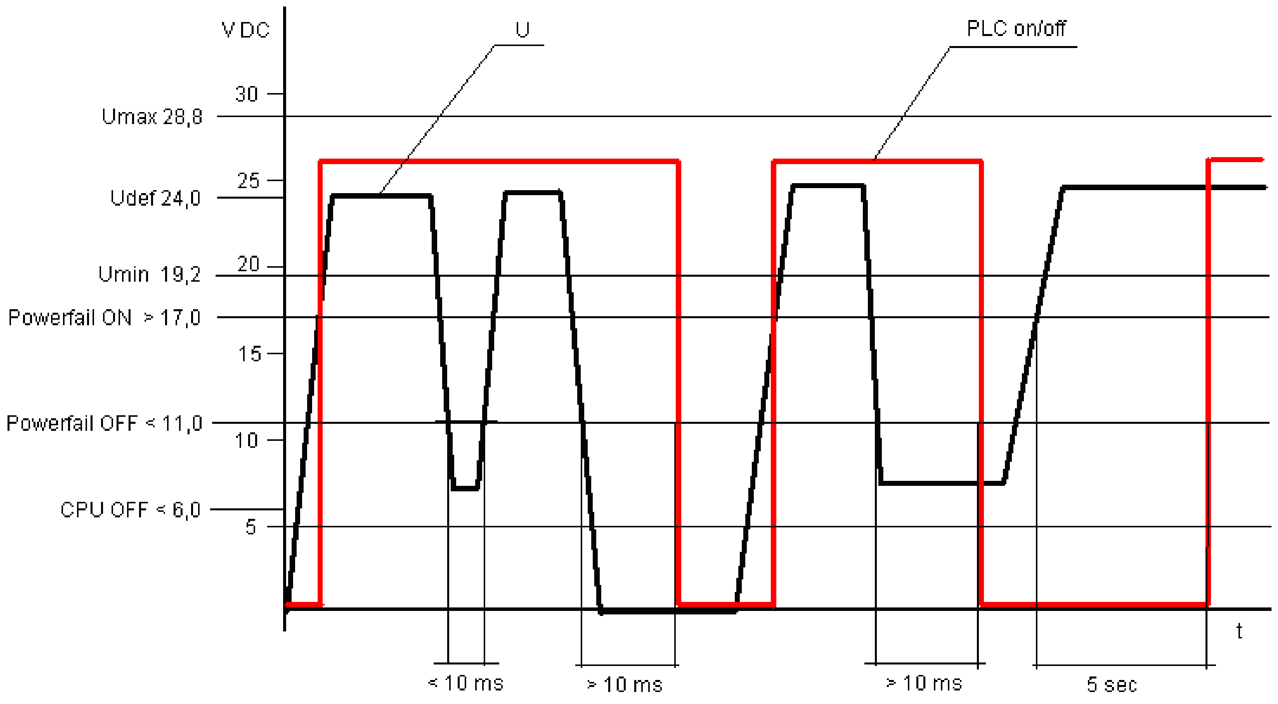
The parameter "Warmstart" allows to the set the behavior of the CPU in case of

- serious errors (class E2) and
- short voltage dips.

If the default setting is used, the CPU changes to STOP mode if a serious error occurs. The CPU is switched off for voltage dips >10ms. The display shows "AC500".

The new settings allow to perform a warmstart of the CPU after a serious error or after short voltage dips or in case of both events.

The following figure shows the behavior of the CPU for different control voltage signals.



Short voltage dips, i.e., the control voltage falls below a value lower than "Powerfail OFF" (<11 VDC) for less than 10 ms, are bridged by the PLC, i.e., the CPU remains on.

If the control voltage is switched off, the CPU remains on for > 10 ms.

If the control voltage is lower than 11 V DC (but > 6 V DC) for longer than 10 ms and then goes back to the normal value, the behavior of the CPU depends on the setting for the parameter "Warmstart". If the parameter is set to "Off", the CPU remains in power fail mode, i.e., it does not restart. A restart of the CPU can only be done by switching the control voltage OFF/ON. If the parameter is set to "On after short voltage dip" or "On after E2 or short voltage dip", the CPU is restarted when the control voltage is greater than 17 V DC for 5 seconds. However, if the control voltage falls once more below 11 V DC within these 5 seconds, the time is restarted. Thus, the control voltage must have a value > 17 V DC for 5 seconds.

Remark 7: Start PERSISTENT %Rsegment.x and End PERSISTENT %Rsegment.x

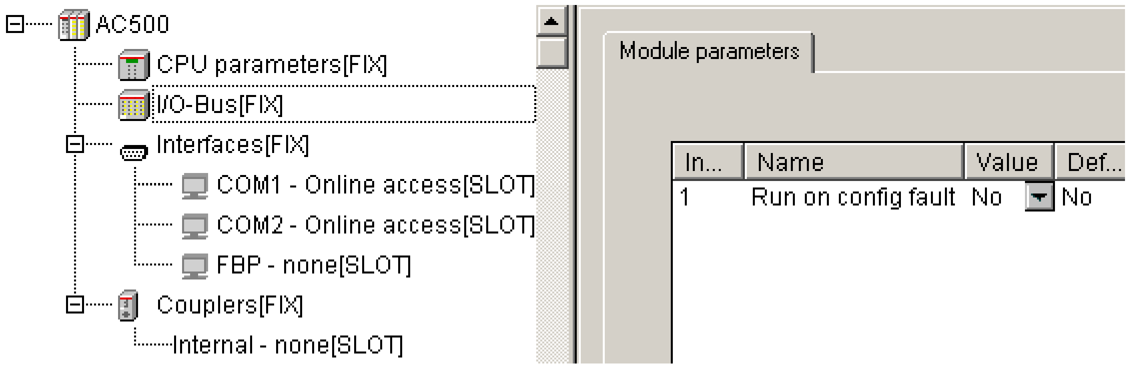
With version V1.2 of PS501 and firmware V1.2.0 the new addressable variables area %Rx.x is available. The parameters "Start PERSISTENT %Rsegment.x" and "End PERSISTENT %Rsegment.x" are used to buffer this area. In the particular segment, "Start PERSISTENT %Rsegment.x" specifies the start byte and "End PERSISTENT %Rsegment.x" the end byte of the area to be buffered.

The new operand area is described in detail in chapter "The addressable PERSISTENT area %Rsegment.x".

3.3 I/O bus configuration

3.3.1 Setting the general I/O bus parameters

Selecting "I/O-Bus" in the configuration tree opens the following configuration window:



In...	Name	Value	Def...
1	Run on config fault	No	No

In the same way as described for the CPU parameters, the general parameters for the CPU's I/O bus can be set in this window. The following parameter can be set:

Parameter	Default value	Value	Meaning
Run on config fault	No	No	In case of a configuration error, the user program is not started.
		Yes	The user program is run independent of a faulty I/O bus configuration.

3.3.2 Inserting input and output modules

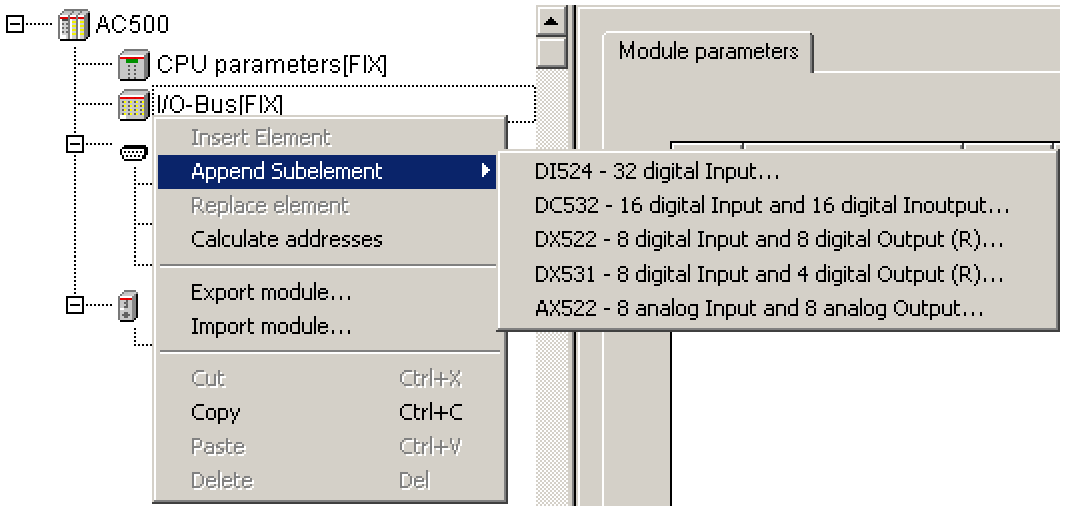
To make the inputs and/or outputs of the input and output modules connected to the I/O bus available in the project, the hardware must be reproduced in the PLC configuration.

Input and output modules connected to the I/O bus of the CPU occupy the I/O following area:

%IB0 .. %IB999 or %QB0 .. %QB999.

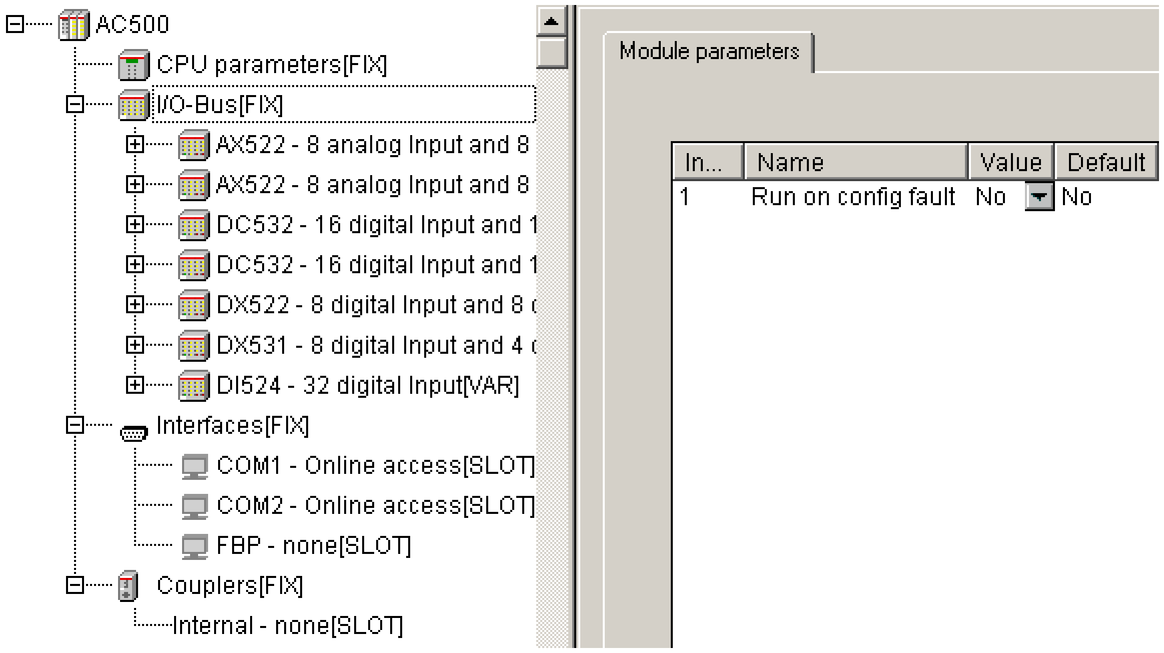
There is no fix assignment between module number and the input/output addresses of the channels.

Right-clicking the "I/O-Bus" element in the configuration tree opens the context menu where you can change the "I/O_Bus" mode. Select "Append Subelement". The sub menu displays all available input and output modules:


AC500
CPU parameters[FIX]
I/O-Bus[FIX]
Module parameters
Insert Element
Append Subelement
Replace element
Calculate addresses
Export module...
Import module...
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Delete Del
DI524 - 32 digital Input...
DC532 - 16 digital Input and 16 digital Inoutput...
DX522 - 8 digital Input and 8 digital Output (R)...
DX531 - 8 digital Input and 4 digital Output (R)...
AX522 - 8 analog Input and 8 analog Output...

Select the desired module depending on its hardware configuration. Repeat this step for all modules. A maximum of 7 input/output modules (10 modules as of V1.2.0) can be appended to the I/O bus.

The following figure shows an example for a configuration with the maximum number of modules:


AC500
CPU parameters[FIX]
I/O-Bus[FIX]
AX522 - 8 analog Input and 8
AX522 - 8 analog Input and 8
DC532 - 16 digital Input and 1
DC532 - 16 digital Input and 1
DX522 - 8 digital Input and 8 (
DX531 - 8 digital Input and 4 (
DI524 - 32 digital Input[VAR]
Interfaces[FIX]
COM1 - Online access[SLOT]
COM2 - Online access[SLOT]
FBP - none[SLOT]
Couplers[FIX]
Internal - none[SLOT]
Module parameters

In...	Name	Value	Default
1	Run on config fault	No	No

If the maximum number of modules (7 modules or 10 modules as of V1.2.0) are appended, the context menu item "Append Subelement" can no longer be selected.

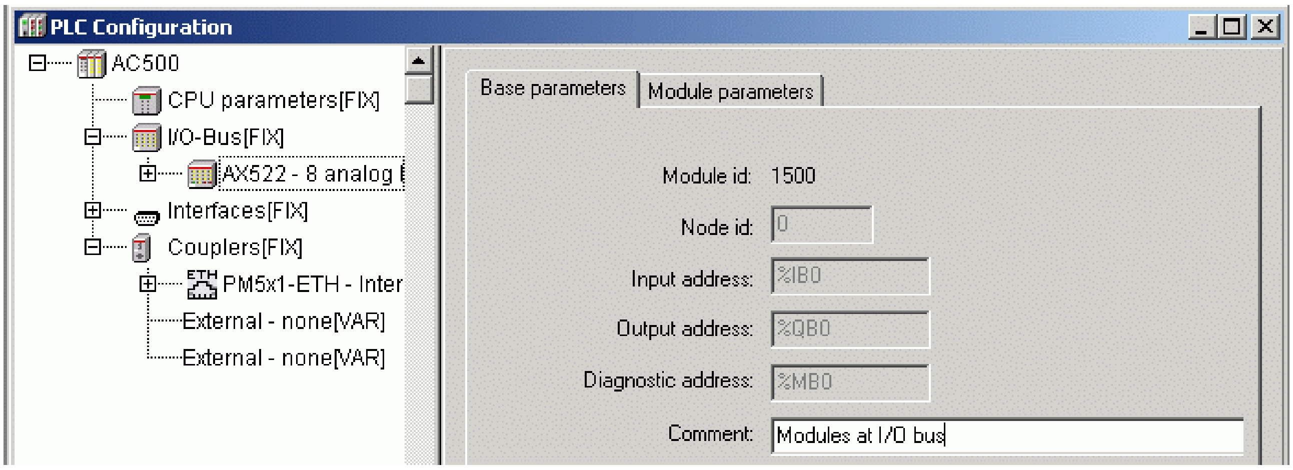
Changing the configuration is possible by deleting modules and inserting or appending new modules.

Note: As of firmware version V1.2.0 and PS501 version V1.2, 10 input/output modules can be appended to the CPU's I/O bus.

3.3.3 Configuring the input and output modules and channels

All inputs and/or outputs of the module are created when inserting the input/output module. In case of digital modules, the channels are provided as WORD, BYTE and BOOL.

If version V1.0 or V1.1 of the Control Builder is used, the module parameters are directly shown when clicking on an input/output module. As of Control Builder version V1.2, the Base parameters tab is opened.



Module id: 1500

Node id: 0

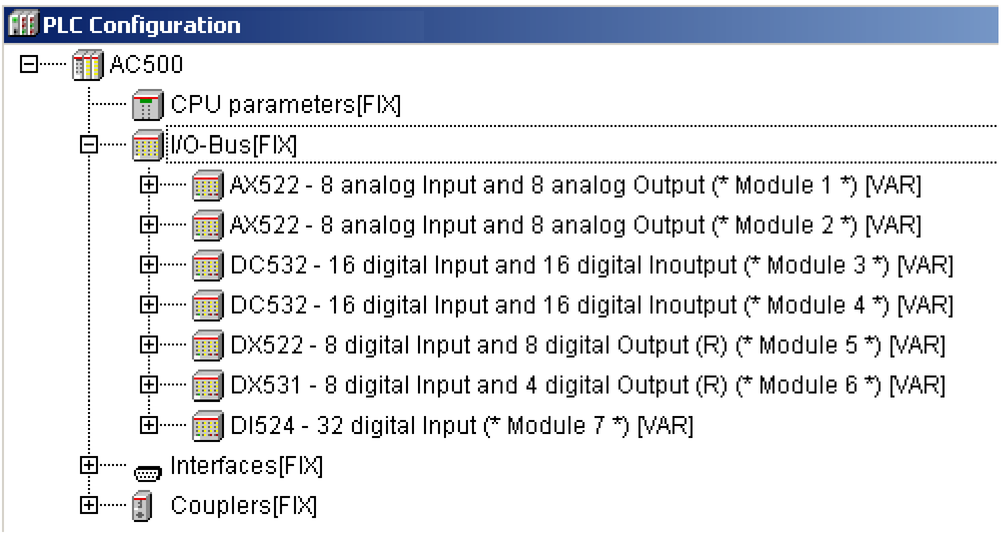
Input address: %IB0

Output address: %QB0

Diagnostic address: %MB0

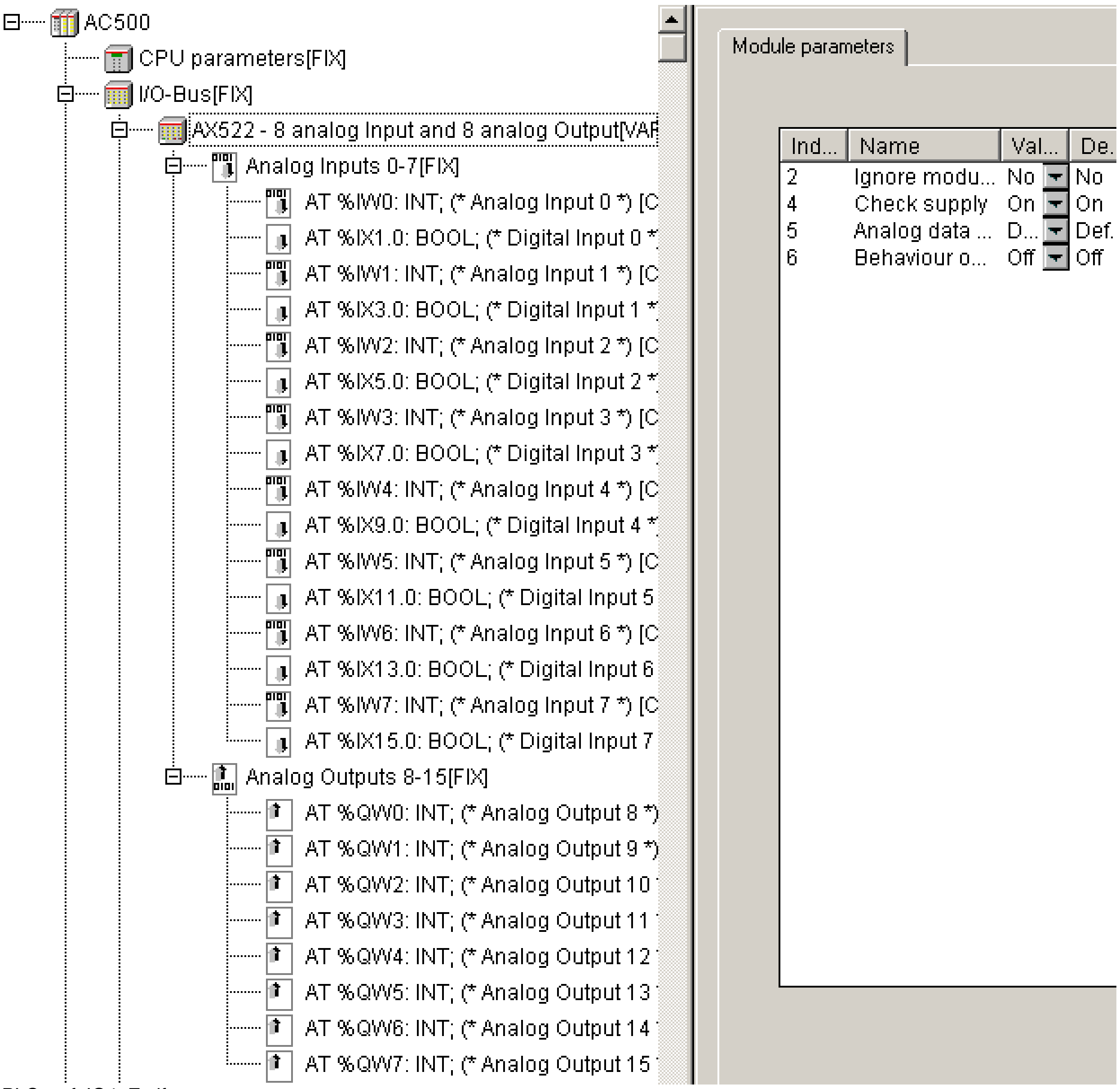
Comment: Modules at I/O bus

A module name can be entered into the Comment field. This name also appears in the tree structure.



```
PLC Configuration
⊟ AC500
    CPU parameters[FIX]
  ⊟ I/O-Bus[FIX]
      ⊞ AX522 - 8 analog Input and 8 analog Output (* Module 1 *) [VAR]
      ⊞ AX522 - 8 analog Input and 8 analog Output (* Module 2 *) [VAR]
      ⊞ DC532 - 16 digital Input and 16 digital Inoutput (* Module 3 *) [VAR]
      ⊞ DC532 - 16 digital Input and 16 digital Inoutput (* Module 4 *) [VAR]
      ⊞ DX522 - 8 digital Input and 8 digital Output (R) (* Module 5 *) [VAR]
      ⊞ DX531 - 8 digital Input and 4 digital Output (R) (* Module 6 *) [VAR]
      ⊞ DI524 - 32 digital Input (* Module 7 *) [VAR]
  ⊞ Interfaces[FIX]
  ⊞ Couplers[FIX]
```

If you expand, for instance, the analog module AX522 and select the module parameters, the following is displayed:



```
⊟ AC500
    CPU parameters[FIX]
  ⊟ I/O-Bus[FIX]
    ⊟ AX522 - 8 analog Input and 8 analog Output[VAR
      ⊟ Analog Inputs 0-7[FIX]
          AT %IW0: INT; (* Analog Input 0 *) [C
          AT %IX1.0: BOOL; (* Digital Input 0 *
          AT %IW1: INT; (* Analog Input 1 *) [C
          AT %IX3.0: BOOL; (* Digital Input 1 *
          AT %IW2: INT; (* Analog Input 2 *) [C
          AT %IX5.0: BOOL; (* Digital Input 2 *
          AT %IW3: INT; (* Analog Input 3 *) [C
          AT %IX7.0: BOOL; (* Digital Input 3 *
          AT %IW4: INT; (* Analog Input 4 *) [C
          AT %IX9.0: BOOL; (* Digital Input 4 *
          AT %IW5: INT; (* Analog Input 5 *) [C
          AT %IX11.0: BOOL; (* Digital Input 5
          AT %IW6: INT; (* Analog Input 6 *) [C
          AT %IX13.0: BOOL; (* Digital Input 6
          AT %IW7: INT; (* Analog Input 7 *) [C
          AT %IX15.0: BOOL; (* Digital Input 7
      ⊟ Analog Outputs 8-15[FIX]
          AT %QW0: INT; (* Analog Output 8 *)
          AT %QW1: INT; (* Analog Output 9 *)
          AT %QW2: INT; (* Analog Output 10
          AT %QW3: INT; (* Analog Output 11
          AT %QW4: INT; (* Analog Output 12
          AT %QW5: INT; (* Analog Output 13
          AT %QW6: INT; (* Analog Output 14
          AT %QW7: INT; (* Analog Output 15
```

Module parameters

Ind...	Name	Val...	De.
2	Ignore modu...	No	No
4	Check supply	On	On
5	Analog data ...	D...	Def.
6	Behaviour o...	Off	Off

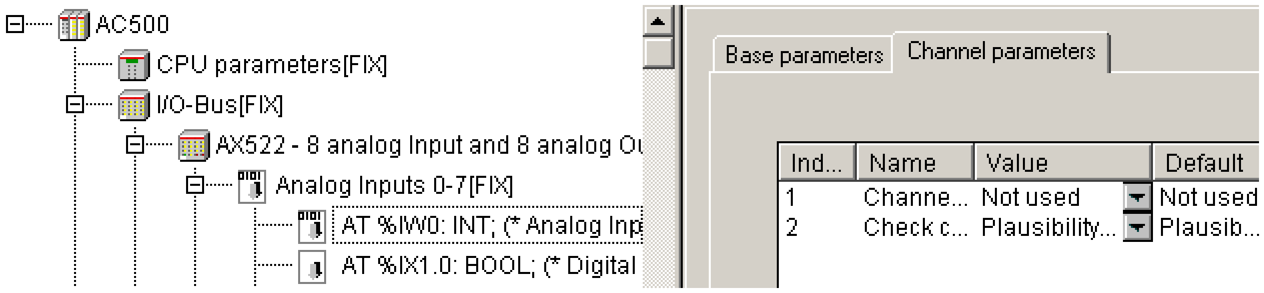
PLCconf_IO4_E.gif

Because the analog inputs can also be configured as digital inputs, bit 0 of each channel is also available as BOOL.

The following settings are possible:

1. The window with the module-specific parameters is displayed by selecting the module in the configuration tree. The parameters differ for the individual modules. For a description of the module parameters refer to the documentation for the input/output modules (parameterization) (see also I/O Device Description / Modules).

   If an input/output module contains channel-related parameters, the following window appears when selecting the corresponding channel in the configuration tree:



AC500
CPU parameters[FIX]
I/O-Bus[FIX]
AX522 - 8 analog Input and 8 analog Ou
Analog Inputs 0-7[FIX]
AT %IW0: INT; (* Analog Inp
AT %IX1.0: BOOL; (* Digital

Base parameters | Channel parameters

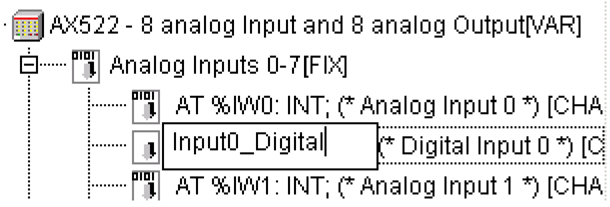
Ind...	Name	Value	Default
1	Channe...	Not used	Not used
2	Check c...	Plausibility...	Plausib...

The parameters differ for the individual modules. For a description of the module parameters refer to the documentation for the input/output modules.

2. The symbolic name of a channel can be entered in front of the string "AT" in the channel declaration.

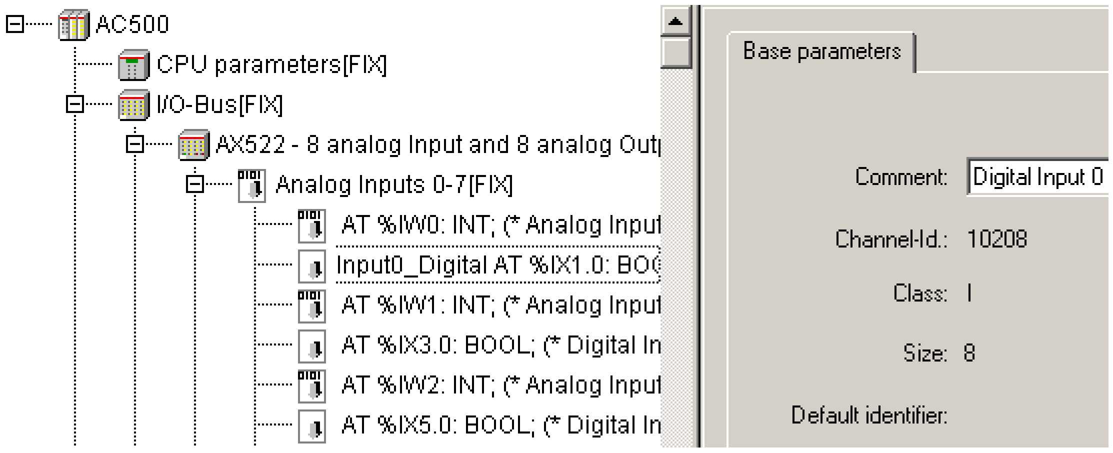
Note: All channels should have a symbolic name and only symbolic names should be used in the program code. If the hardware configuration has changed or if you want to download the project to a PLC with another hardware configuration and thus the PLC configuration has to be changed, the addresses of the inputs and outputs can change. In case of symbolic programming (i.e., symbolic names are used), the program code does not have to be changed.

Example how to enter a symbolic name:



AX522 - 8 analog Input and 8 analog Output[VAR]
Analog Inputs 0-7[FIX]
AT %IW0: INT; (* Analog Input 0 *) [CHA
Input0_Digital (* Digital Input 0 *) [C
AT %IW1: INT; (* Analog Input 1 *) [CHA

3. For each channel, a comment can be entered into the field "Comment" in the "Base parameters" tab.



AC500
CPU parameters[FIX]
I/O-Bus[FIX]
AX522 - 8 analog Input and 8 analog Out
Analog Inputs 0-7[FIX]
AT %IW0: INT; (* Analog Input
Input0_Digital AT %IX1.0: BO
AT %IW1: INT; (* Analog Input
AT %IX3.0: BOOL; (* Digital In
AT %IW2: INT; (* Analog Input
AT %IX5.0: BOOL; (* Digital In

Base parameters

Comment: Digital Input 0
Channel-Id.: 10208
Class: I
Size: 8
Default identifier:

3.3.4 Module parameter "Ignore module" of S500 I/O devices

All S500 I/O devices have the module parameter "Ignore module". This parameter allows to set whether the I/O device specified in the PLC configuration is considered or not when checking the configuration data.

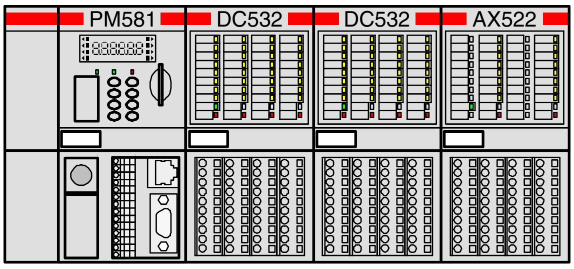
The parameter setting No (default setting) requires that the device is physically available.

If the parameter is set to Yes, the device must not be connected!

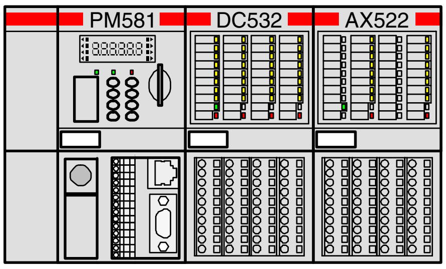
Thus, it is for example possible to create a project for machines with different hardware configuration and to exclude unnecessary input/output devices from checking by setting the parameter Ignore module to TRUE.

Example:

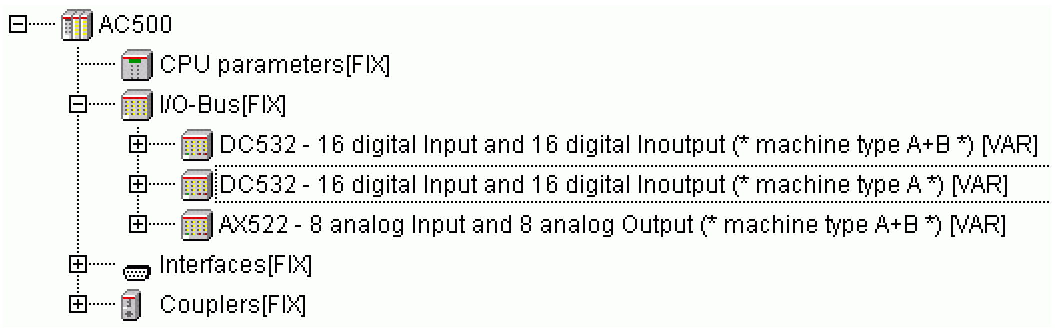
In full installation (type A), a machine shall be controlled with an AC500 with the following hardware configuration: CPU PM581 + 2xDC532 + 1xAX522



For a variant (type B) of the machine, the second DC532 is not required. This results in the following PLC hardware configuration: CPU PM581 + 1xDC532 + 1xAX522



The PLC configuration is identical for both machines:



```
⊟ AC500
   CPU parameters[FIX]
   ⊟ I/O-Bus[FIX]
      ⊞ DC532 - 16 digital Input and 16 digital Inoutput (* machine type A+B *) [VAR]
      ⊞ DC532 - 16 digital Input and 16 digital Inoutput (* machine type A *) [VAR]
      ⊞ AX522 - 8 analog Input and 8 analog Output (* machine type A+B *) [VAR]
   ⊞ Interfaces[FIX]
   ⊞ Couplers[FIX]
```

In the project for machine type B, the module parameter Ignore module is set to TRUE for the second DC532. Thus, all inputs and outputs have the same addresses.

A further advantage of this parameter is that, for example, not all devices must be available for test purposes.

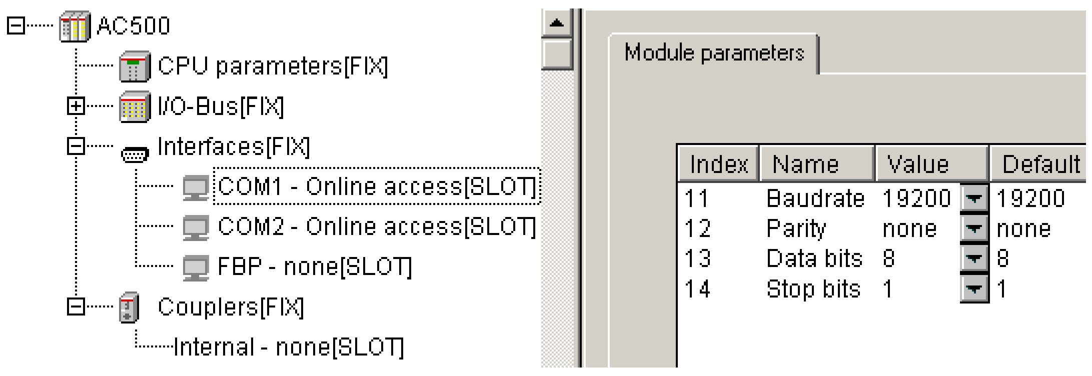
3.4 Configuration of the serial interfaces (Interfaces / COM1 and COM2)

The AC500 CPU is equipped with the two interfaces COM1 and COM2 which can be operated as RS 232 and RS 485.

Note: RS 485 operation of an interface is only possible, if the parameter "RTS control" is set to "telegram".

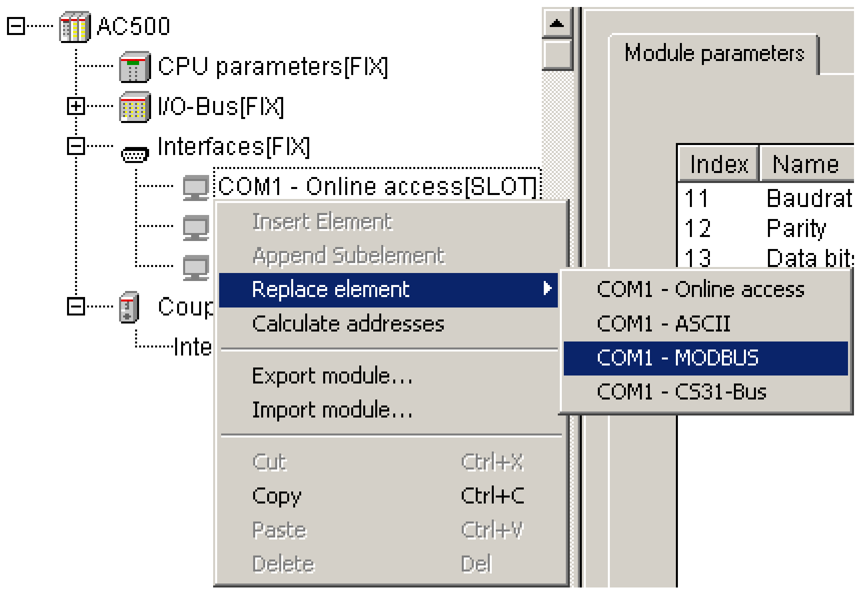
3.4.1 Setting the protocol of the serial interfaces

By default, the serial interfaces are set to 'Online access', i.e., the access is done with help of the Control Builder.



Index	Name	Value	Default
11	Baudrate	19200	19200
12	Parity	none	none
13	Data bits	8	8
14	Stop bits	1	1

The protocol of the serial interfaces can be changed by right-clicking the interface 'COM1' or 'COM2' in the configuration tree and selecting the context menu item 'Replace element'.



That means, the interface protocol is directly set in the PLC configuration. No block (such as MODINIT, COMINIT) is required.

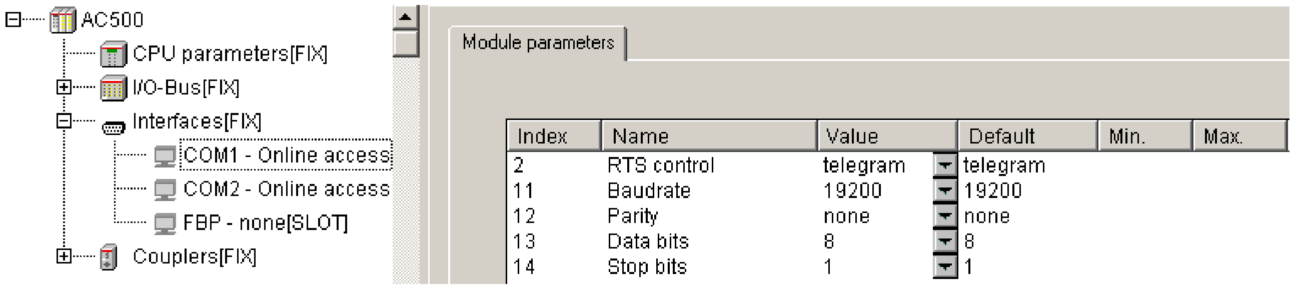
The serial interface settings can be read in online mode using the PLC browser commands "com settings" and "com protocols". Chapter "AC500-specific PLC browser commands" contains a description of these commands.

3.4.2 The setting 'COMx - Online access'

If 'COMx - Online access' is selected, the interface parameters are set to the following fixed values:

Baudrate=19200 Baud, Stop bit=1, Parity=none, Data bits=8

Note: As of firmware version V1.1.7 and Control Builder version V1.2, the parameter "RTS control" is set to "telegram". This also allows programming via RS 485 (for example using an according converter).



Index	Name	Value	Default	Min.	Max.
2	RTS control	telegram	telegram		
11	Baudrate	19200	19200		
12	Parity	none	none		
13	Data bits	8	8		
14	Stop bits	1	1		

The parameters are read-only (not editable).

The serial interface settings must match the settings for the serial gateway driver in the Control Builder (see also Programming and Testing / Serial Driver).

3.4.3 The setting 'COMx - ASCII'

With the selection "ASCII", the initialization of the serial interface is done for the "free protocol", i.e., all interface parameters can be set and any protocol can be realized.

Sending and receiving data is done by means of the blocks COM_SEND and COM_REC (contained in library ASCII_AC500_V10.LIB). A detailed description of these blocks can be found in the ASCII_AC500_V10.LIB documentation.

Caution: To be able to receive data using the block COM_REC, a buffer of the size **272 bytes** must be available (for example abyRecData : ARRAY[0..271] OF BYTE).
This is also required if only short telegrams are received.

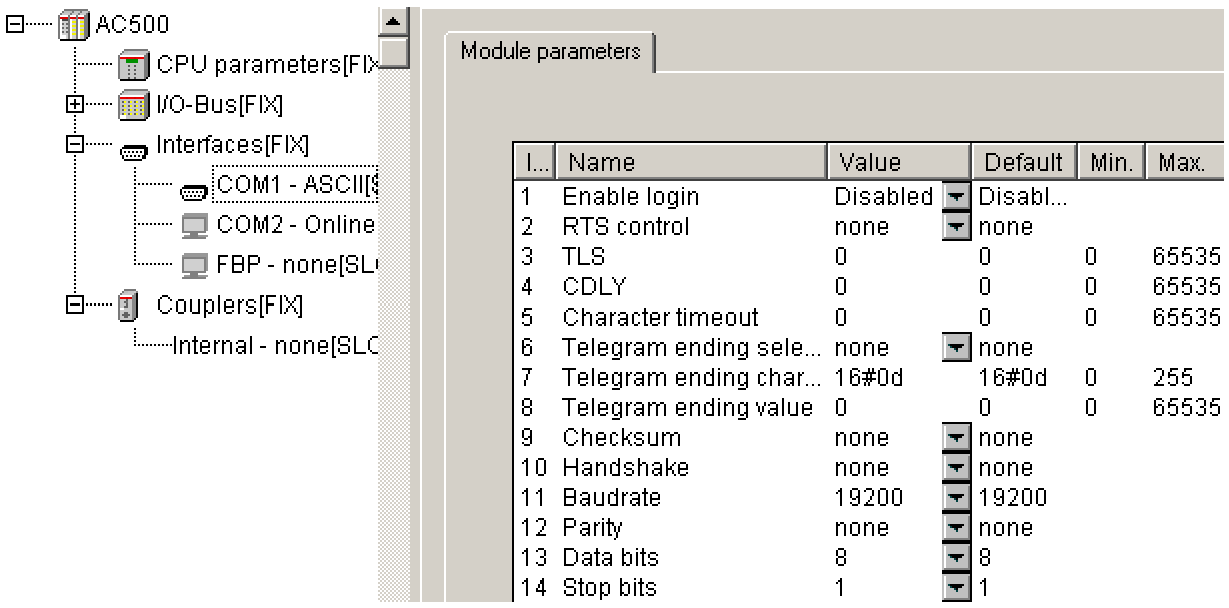
The operating system provides a total of 32 buffers with 272 bytes each for the transmission and reception of data. If the PLC is in STOP mode (= pause) or the input EN at the block COM_REC is set to FALSE or the block is not called, these buffers run full.

If the block COM_REC is called again (with EN:=TRUE) before all buffers are used, the data received meanwhile are made available.

If all buffers were full, the error Invalid handle with ERR=TRUE and ERNO=16#2001=8193 is reported for one cycle. After this the reception is reset.

The reception is always reset after a download or the command Online/Reset.

Selecting "ASCII" displays the following window:



I...	Name	Value	Default	Min.	Max.
1	Enable login	Disabled	Disabl...		
2	RTS control	none	none		
3	TLS	0	0	0	65535
4	CDLY	0	0	0	65535
5	Character timeout	0	0	0	65535
6	Telegram ending sele...	none	none		
7	Telegram ending char...	16#0d	16#0d	0	255
8	Telegram ending value	0	0	0	65535
9	Checksum	none	none		
10	Handshake	none	none		
11	Baudrate	19200	19200		
12	Parity	none	none		
13	Data bits	8	8		
14	Stop bits	1	1		

The parameters define how the serial interface will be initialized. The parameters can be grouped. They are used to initialize the following functions:

- Monitoring the programming login:
  Enable login
- Modem control and RS485:
  RTS control, TLS, CDLY
- Recognition of telegram ending for reception:
  Character timeout, Telegram ending selection, Telegram ending value, Telegram ending character
- Checksum
- Transmission parameters:
  Baudrate, Parity, Data bits, Stop bits

The following settings are possible:

Parameter	**Default value**	**Value**	**Meaning**
Enable login **see remark 1**	Disabled	Disabled	There is no check with regard to the Control Builder login telegram.
		Enabled	Telegrams received are checked with regard to the Control Builder login sequence. If the sequence is detected, the protocol setting is changed to 'Online access'. -> available as of firmware 1.2.0 and PS501 V1.2
RTS control **see remark 2**	none	None	No RTS control (direction control)
		telegram	RTS control activated (absolutely necessary for RS 485!)
TLS **see remark 2**	0	0...65535	Carrier lead time in [ms] (TLS > CDLY)
CDLY **see remark 2**	0	0...65535	Carrier delay time in [ms] (CDLY <= TLS)
Character timeout	0	0...65535	Character timeout in characters (must be 0 if Telegram ending selection = Character timeout)

see remark 3			
Telegram ending selection **see remarks 3 and 4**	none	none	No telegram ending identifier
		String (check receive)	2 characters, e.g. <CR><LF> (16#0d, 16#0a -> 16#0d0a) in parameter "Telegram ending value"
		Telegram length	Telegram ending identifier set by telegram length
		Duration	Telegram ending identifier set by time
		Character timeout	Telegram ending identifier set by character timeout
Telegram ending character **see remark 3**	16#0d	0...255	Up to version V1.1.x: Telegram ending character
	0	0...1	As of version V1.2.0: Number of end characters in case of telegram ending selection "String"
Telegram ending value **see remark 3**	0	0...65535	Up to version V1.1.x: Telegram ending identifier value for settings "Duration" and "Character timeout"
	0	0...65535	As of version V1.2.0: Telegram ending identifier value for settings "Duration", "Character timeout" and "String"
Checksum **see remark 4**	none	None	No checksum
		CRC8	CRC8 checksum -> available as of firmware V1.2.0
		CRC16	CRC16 checksum (Motorola format) -> available as of firmware V1.2.0
		LRC	Add all values to byte (ignore overflow), result multiplied by -1 -> available as of firmware V1.2.0
		ADD	Add all values to byte (ignore overflow) -> available as of firmware V1.2.0 and PS501 V1.2
		CS31	CS31 bus checksum -> available as of firmware V1.2.0 and PS501 V1.2
		CRC8-FBP	CRC8 FBP field bus neutral protocol -> available as of firmware V1.2.0 and PS501 V1.2
		XOR	XOR all values to byte (ignore overflow) -> available as of firmware V1.2.0 and PS501 V1.2
		CRC16 (Intel)	Like CRC16, result swapped -> available as of firmware V1.2.0 and PS501 V1.2
Handshake	none	None	No handshake
		RTS/CTS	Hardware handshake
		XON/XOFF	Not yet implemented
		3964R master	Not yet implemented
		3964R slave	Not yet implemented
Baudrate	19200	300 1200 4800 9600 14400 19200 38400 57600 115200 125000 187500	Character length in bits/s

Parity	none	None	No parity check
		Odd	Odd parity
		Even	Even parity
		Mark	Parity bit := TRUE
		Space	Parity bit := FALSE
Data bits	8	5, 6, 7, 8	Character length in bits/character
Stop bits	1	1, 2	Number of stop bits

Remark 1: Enable login

This parameter is available as of firmware version V1.2.0 and PS501 V1.2!

If "Enable login" is set to Yes, all received telegrams are checked with regard to the CoDeSys login service.

⚠ **Caution:** It is recommended to activate the automatic login detection only for those projects for which this function is absolutely required because it slows down communication via the serial interface and also influences the PLC performance.

If the connection is directly made via RS 232, a login telegram will only be detected if the same parameters as used by CoDeSys (Baudrate=19200 Baud, Stop bits=1, Parity=None, Data bits=8 Bit) are set when initializing the interface.

The same applies if the connection is made via RS 232/RS 485 interface converters. The login telegram can only be detected, if the initialization parameters have the same values as the parameters set in CoDeSys. Because for such an application usually more than one device are connected to the RS 485 transmission line, the following has to be observed additionally:

The CoDeSys login telegram does not contain a device address. Thus, the service is first identified by all devices connected to the RS 485 transmission line that can be programmed using CoDeSys and the interface of which is able to read the login telegram (interface with Enable login=Yes). Due to this, telegram collisions can occur during the subsequent acknowledgement of the login request by these devices, resulting in an interruption of the communication.

If the connection between CoDeSys and the PLC is established via modem, the communication is not influenced by the interface parameters set in the PLC configuration. The parameter values required for the modem used have to be set. Once the initialization is completed, the mode processes the received telegrams according to the parameter settings. Also the assignment between login request and an individual PLC is guaranteed because the connection is established using the modem's phone number or MSN.

The login with CoDeSys first causes a reinitialization of the interface. All blocks accessing this interface are locked during the online session, i.e., they do not perform any function. During this period the block outputs have the following values:

DONE = FALSE
ERR = TRUE
ERNO = PROTOCOL_PROTECTED = 16#301F = 12319

The blocks will be re-activated after the logout by CoDeSys.

The login monitoring for an interface is only done if CoDeSys is not already logged in via another interface (Ethernet, ARCNET or other COM).

Remark 2: Usage of modems

The ASCII protocol considers the special properties of modems, interface converters and repeaters. If these devices are used at a serial interface operated in 'free mode', the compression mechanism possibly supported by these devices has to be deactivated. For detailed information, please refer to the operation manual of the used device.

Some repeaters, modems or interface converters require a control signal in order to set the transfer direction. The direction control can be enabled or disabled via the input RTSCTRL.

Various devices additionally require a lead time to stabilize their carrier signal. These devices can only transfer data in send direction after this time has elapsed. This carrier lead time can be set via the input TLS.

Additionally, for some devices it is necessary to sustain the carrier signal in send direction for some time after data transfer is completed. Only if this time has elapsed, the complete transfer of a telegram is ensured and the devices are ready for data transfer in opposite direction. This carrier delay time can be set via the input CDLY.

Note: Carrier lead time (TLS) and carrier delay time (CDLY) must be adjusted for all communication devices connected to the same transmission line.
The times are only considered for RTS control = telegram.

Remark 3: Telegram ending identifier

The telegram ending identifier is set using the parameters Character timeout, Telegram ending selection, Telegram ending character and Telegram ending value.

Character silent time monitoring:
Monitoring of the character timeout can be set for all possible telegram ending settings (except Character timeout).

If the parameter "Character timeout" = 0, no character timeout monitoring is done.

With "Character timeout" > 0 the character timeout monitoring is activated.

The character silent time is defined in number of characters. The number of characters and the interface parameters (Baudrate, Parity, Data bits and Stop bits) are used to calculate the silent time.

Example: Baudrate=9600 Baud, Parity=none, Data bits=8, Stop bits=1, Character timeout=3

This results in a frame of 10 bits/character:
1 start bit + 8 data bits + 0 parity bit + 1 stop bit

Character silent time = 1000 x Character timeout x Frame / Baudrate [ms]
Character timeout = 1000 x 3 x 10 / 9600 = 3.125 ms ~ 4 ms.

If the time between the reception of two characters exceeds the character silent time, the reception is aborted with an error and the characters received up to this moment are made available.

The following parameter combinations are possible:

Character timeout	**Telegram ending selection**	**Telegram ending character**	**Telegram ending value**	**Description**
Character timeout **see remark on character silent time monitoring**	**Type of telegram ending identifier**	**Telegram ending character** **- = ignored**	**Telegram ending value** **- = ignored**	
Number of characters 0 or >0	None	-	-	No telegram ending identifier, i.e., the characters received since last call are provided. The maximum number of characters is limited to 256.
Number of characters 0 or >0	String (check receive)	Number of telegram ending characters 1 or 2	2 characters (for example 16#0d0a)	According to value set for "Telegram ending character", it is checked for 1 or 2 ending characters. The ending character(s) is (are) not passed, i.e., they are not contained in DATA area.
	1	1	16#0d = 13dec = <CR>	After reception of 16#0d, telegram received is reported.
	2	2	16#0d0a = 3338dec = <CR><LF>	After reception of 16#0d and subsequently 16#0a, telegram received is reported.
Number of characters 0 or >0	Telegram length	-	Number of characters >0 and <=256	Telegram received is reported once the number of characters defined in "Telegram ending value" is received.
Number of characters 0 or >0	Duration	-	Time in [ms]	Telegram received is reported once the time set for "Telegram ending value" (in [ms]) is elapsed. The time starts with the first FALSE -> TRUE edge at input EN of the receive block COM_REC.
0	Character timeout	-	Number of characters >0 and <=256	The number of characters set for "Telegram ending value" and the interface parameters (Baudrate, Parity, Data bits and Stop bits) are used to calculate the silent time. Telegram received is reported if the silent time between two characters is >= the calculated silent time.

⚠ **Caution:** The setting for the telegram ending selection "String" has been changed for firmware version V1.2.x and Control Builder version V1.2. This setting is not compatible with the setting in the firmware versions V1.0.x and V1.1.x and Control Builder versions V1.0 and V1.1.

Up to version V1.2.x, the telegram ending character was set with the parameter "Telegram ending character", the parameter "Telegram ending value" was ignored. Only one telegram ending character could be set.
Thus, the user program has to be changed accordingly when updating to V1.2.x!

Remark 4: Checksum:

The parameter "Checksum" takes effect as of CPU firmware version V1.2.0.

Sending with block COM_SEND:
With "Checksum" <> none, the selected checksum is appended when sending. If the parameter "Telegram ending selection" is set to "String (check receive)", the checksum of the ending character(s) is entered. The character(s) is (are) appended according to the inputs END_LEN and END_CH of the block COM_SEND.

Receiving with block COM_REC:
With "Checksum" <> none, the selected checksum is checked during reception. If the parameter "Telegram ending selection" is set to "String (check receive)", the checksum of the ending character(s) is expected.
The ending character(s) and the checksum are not output, i.e., they are not contained in the DATA area.

A telegram should look as follows:

Data 0	**Data 1**	**Data 2**	**..**	**Data n**	**Check 1**	**[Check 2]**	**End 1**	**[End 2]**

The values enclosed in [] are only relevant for 16 bit checksum or 2 ending characters.
At the blocks COM_SEND and COM_REC, the area addressed via the input DATA contains the following values:

Data 0	**Data 1**	**Data 2**	**..**	**Data n**

Example:

Setting in PLC configuration:
"Telegram ending selection" = String
"Telegram ending character" = 2
"Telegram ending value" = 16#0d0a
"Checksum" = CRC16 (i.e., Motorola format)

Send with COM_SEND:
LEN = n+1
END_LEN = 2
END_CH = 16#0d0a

The area addressed via input **DATA** contains the following data:

Data 0	**Data 1**	**Data 2**	**..**	**Data n**

The following data are sent via the interface:

Data 0	**Data 1**	**Data 2**	**..**	**Data n**	**CRC16 high**	**CRC16 low**	**16#0d**	**16#0a**

Reception with COM_REC:
The interface receives the following telegram:

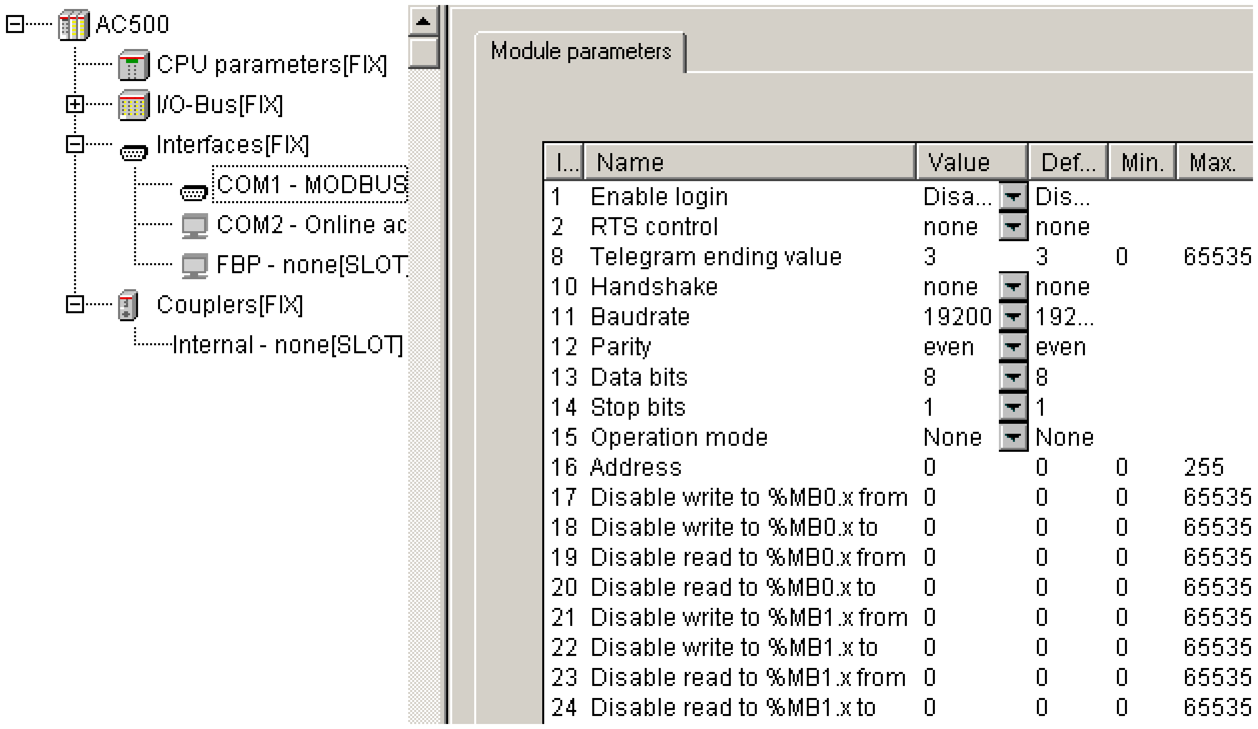
Data 0	**Data 1**	**Data 2**	**..**	**Data n**	**CRC16 high**	**CRC16 low**	**16#0d**	**16#0a**

The following data are written to the area addressed via DATA:

Data 0	**Data 1**	**Data 2**	**..**	**Data n**

3.4.4 The setting 'COMx - Modbus'

For the protocol setting 'MODBUS', the following window is displayed:



I...	Name	Value	Def...	Min.	Max.
1	Enable login	Disa...	Dis...		
2	RTS control	none	none		
8	Telegram ending value	3	3	0	65535
10	Handshake	none	none		
11	Baudrate	19200	192...		
12	Parity	even	even		
13	Data bits	8	8		
14	Stop bits	1	1		
15	Operation mode	None	None		
16	Address	0	0	0	255
17	Disable write to %MB0.x from	0	0	0	65535
18	Disable write to %MB0.x to	0	0	0	65535
19	Disable read to %MB0.x from	0	0	0	65535
20	Disable read to %MB0.x to	0	0	0	65535
21	Disable write to %MB1.x from	0	0	0	65535
22	Disable write to %MB1.x to	0	0	0	65535
23	Disable read to %MB1.x from	0	0	0	65535
24	Disable read to %MB1.x to	0	0	0	65535

The following settings are possible:

Parameter	Default value	Value	Meaning
Enable login	Disabled	Disabled	There is no check with regard to the Control Builder login telegram.
		Enabled	Telegrams received are checked with regard to the Control Builder login sequence. If the sequence is detected, the protocol setting is changed to 'Online access'. -> available as of firmware V1.2.0
RTS control	None	None	No RTS control
		Telegram	RTS control for telegram activated -> available as of firmware V1.2.0
TLS	0	0...65535	Carrier lead time in [ms] or characters (TLS > CDLY) -> available as of firmware V1.2.0
CDLY	0	0...65535	Carrier delay time in [ms] or characters (CDLY <= TLS) -> available as of firmware V1.2.0
Telegram ending value	3	0...65535	Number of characters for character timeout
Handshake	None	None	No flow control
		RTS/CTS	Hardware handshake -> available as of firmware V1.2.0
		XON/XOFF	Software handshake -> Not yet implemented
Baudrate	19200	300 1200 4800 9600 14400	Character length in bits/s

		19200 38400 57600 115200 125000 187500	
Parity	Even	None	No parity
		Odd	Odd parity
		Even	Even parity
		Mark	Parity bit := TRUE
		Space	Parity bit := FALSE
Data bits	8	5, 6, 7, 8	Number of data bits, 5 to 8
Stop bits	1	1, 2	Number of stop bits, 1 or 2
Operation mode	None	None	None
		Master	Master
		Slave	Slave
Address	0	0...255	Address for Modbus slave
Disable write to %MB0.x from	0	0...65535	Disable write access for segment 0 starting at %MB0.x
Disable write to %MB0.x to	0	0...65535	Disable write access for segment 0 up to %MB0.x
Disable read to %MB0.x from	0	0...65535	Disable read access for segment 0 starting at %MB0.x
Disable read to %MB0.x to	0	0...65535	Disable read access for segment 0 up to %MB0.x
Disable write to %MB1.x from	0	0...65535	Disable write access for segment 1 starting at %MB1.x
Disable write to %MB1.x to	0	0...65535	Disable write access for segment 1 up to %MB1.x
Disable read to %MB1.x from	0	0...65535	Disable read access for segment 1 starting at %MB1.x
Disable read to %MB1.x to	0	0...65535	Disable read access for segment 1 up to %MB1.x

The selection "COMx - MODBUS" sets the serial interface x to the Modbus RTU protocol (see also Modbus protocol).

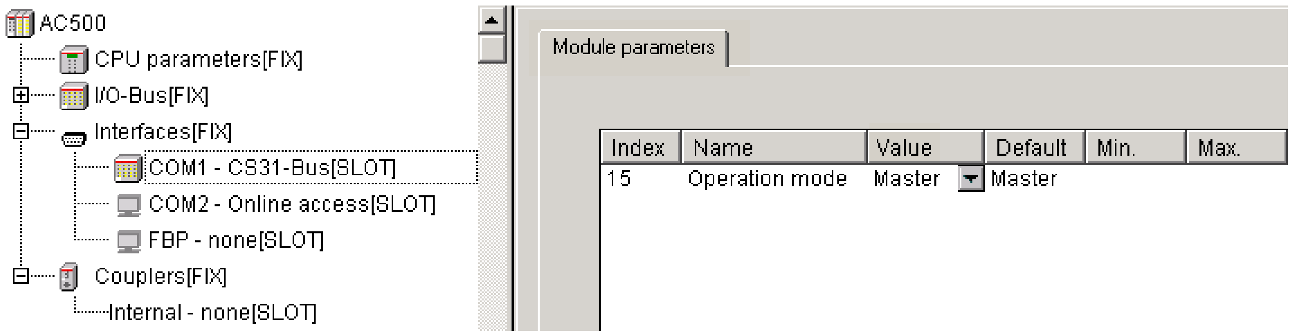
For **Modbus slave operation**, an area without read and/or write access can be set in the segments %M0.x and %M1.x. Reading/writing is disabled beginning at the set address and is valid up to the set end address (inclusive).

Note: The parameter "Data bits" always has to be set to 8 for Modbus.

3.4.5 The setting 'COM1 - CS31 Bus'

If the protocol 'CS31-Bus' is selected for the interface COM1, the interface is definitely set as CS31 bus master.

COM2 cannot be used as CS31 bus interface.

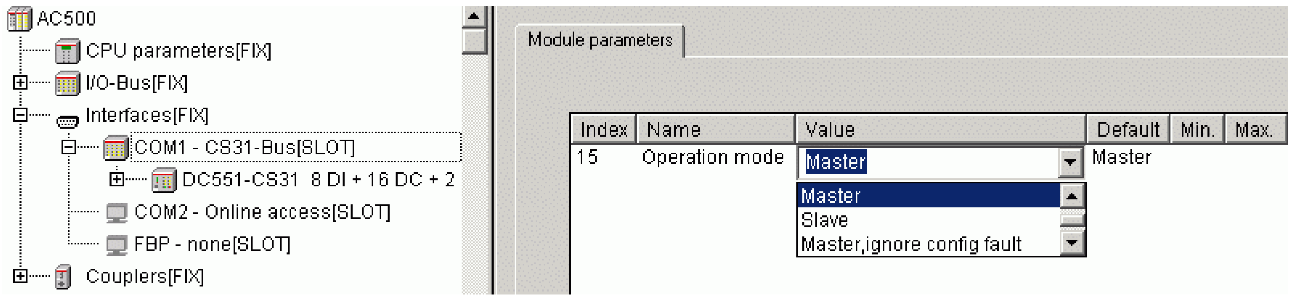

```
AC500
  CPU parameters[FIX]
  I/O-Bus[FIX]
  Interfaces[FIX]
    COM1 - CS31-Bus[SLOT]
    COM2 - Online access[SLOT]
    FBP - none[SLOT]
  Couplers[FIX]
    Internal - none[SLOT]
```

Module parameters

Index	Name	Value	Default	Min.	Max.
15	Operation mode	Master	Master		

As of AC500 firmware version V1.2.0 and Control Builder version V1.2, the parameter "Operation mode" can be set to "Master" (default value) and "Master, ignore config fault" for the CS31.

Note: The settings "Slave" or "Slave, ignore config fault" are not allowed for the "CS31 bus" protocol. The default value "Master" is applied if these values are set.


```
AC500
  CPU parameters[FIX]
  I/O-Bus[FIX]
  Interfaces[FIX]
    COM1 - CS31-Bus[SLOT]
      DC551-CS31 8 DI + 16 DC + 2
    COM2 - Online access[SLOT]
    FBP - none[SLOT]
  Couplers[FIX]
```

Module parameters

Index	Name	Value	Default	Min.	Max.
15	Operation mode	Master	Master		

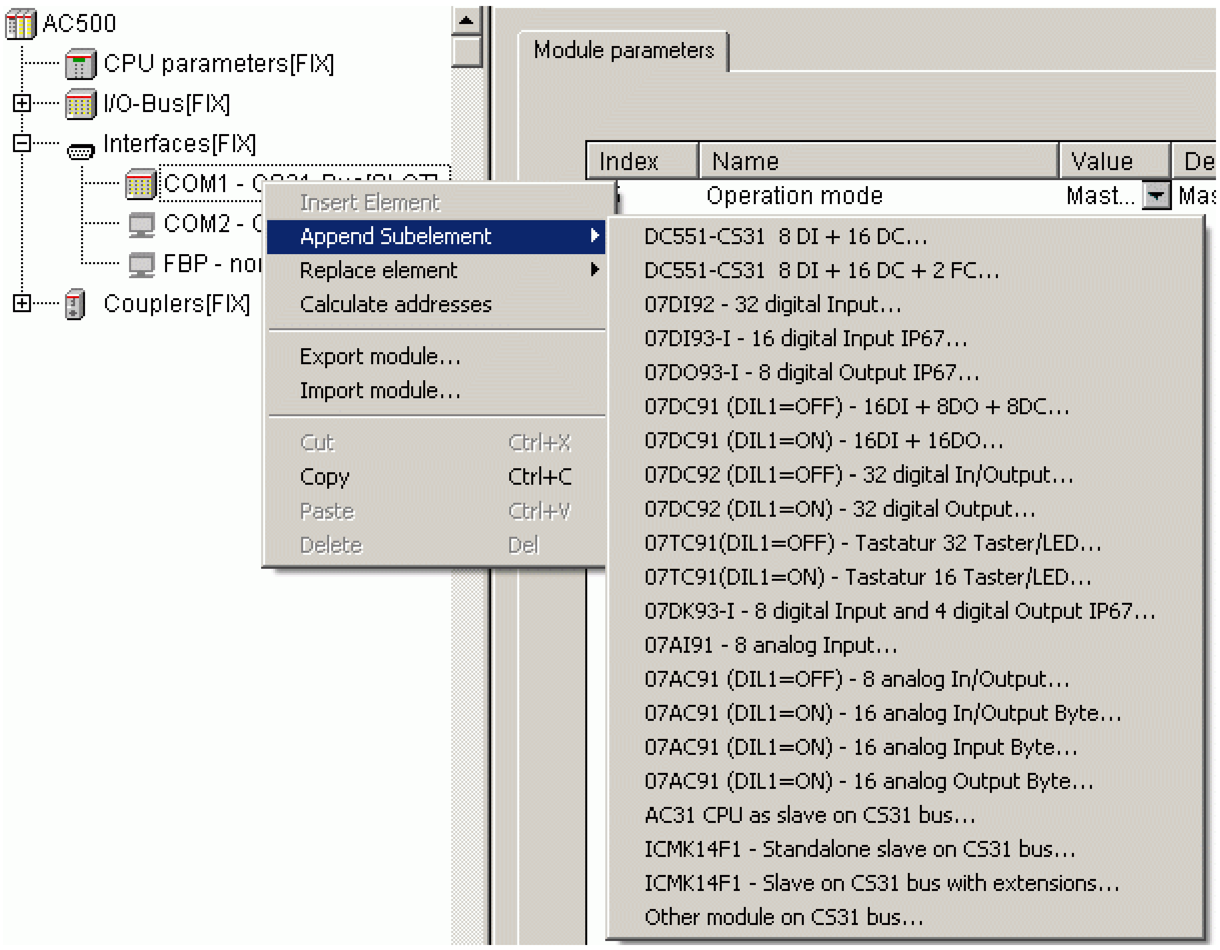
The parameter "Operation mode" influences the beginning of the inputs/outputs update and the start of the user program.

Setting "Master" (default):

Setting "Master, ignore config fault":

To make the inputs and/or outputs of the input and output modules connected to the CPU available in the project, the hardware **must** be reproduced in the PLC configuration.

Right-clicking the "COM1 - CS31-Bus" element in the configuration tree opens the context menu where you can change the module "COM1". Select "Append Subelement". The sub menu displays all input and output modules available for the CS31 bus:


AC500
CPU parameters[FIX]
I/O-Bus[FIX]
Interfaces[FIX]
COM1 - CS31-Bus[SLOT]
COM2 - C
FBP - no
Couplers[FIX]
Insert Element
Append Subelement
Replace element
Calculate addresses
Export module...
Import module...
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Delete Del
DC551-CS31 8 DI + 16 DC...
DC551-CS31 8 DI + 16 DC + 2 FC...
07DI92 - 32 digital Input...
07DI93-I - 16 digital Input IP67...
07DO93-I - 8 digital Output IP67...
07DC91 (DIL1=OFF) - 16DI + 8DO + 8DC...
07DC91 (DIL1=ON) - 16DI + 16DO...
07DC92 (DIL1=OFF) - 32 digital In/Output...
07DC92 (DIL1=ON) - 32 digital Output...
07TC91(DIL1=OFF) - Tastatur 32 Taster/LED...
07TC91(DIL1=ON) - Tastatur 16 Taster/LED...
07DK93-I - 8 digital Input and 4 digital Output IP67...
07AI91 - 8 analog Input...
07AC91 (DIL1=OFF) - 8 analog In/Output...
07AC91 (DIL1=ON) - 16 analog In/Output Byte...
07AC91 (DIL1=ON) - 16 analog Input Byte...
07AC91 (DIL1=ON) - 16 analog Output Byte...
AC31 CPU as slave on CS31 bus...
ICMK14F1 - Standalone slave on CS31 bus...
ICMK14F1 - Slave on CS31 bus with extensions...
Other module on CS31 bus...
Module parameters

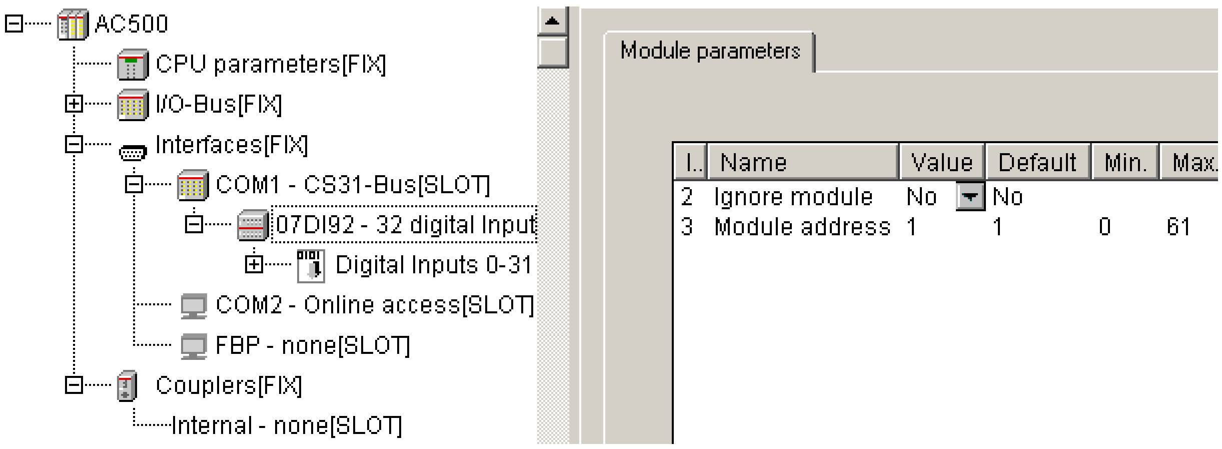
Index	Name	Value	De
	Operation mode	Mast...	Mas

Select the desired input/output module.

A maximum of 31 modules (slaves) can be connected to the CS31 bus. Please note that for a module containing digital and analog expansions two modules are registered on the CS31 bus.

The configuration is described using the digital input module 07DI92 as an example. Once the module is inserted, the inputs and outputs of the module are available. In the 'Module parameters' window, set the 'Module address' to the module's hardware address (this is the address defined at the module with DIL switches).

Note: For AC31 CPUs used as CS31 slave, the address is set via software.



I.	Name	Value	Default	Min.	Max.
2	Ignore module	No	No		
3	Module address	1	1	0	61

The module address has to be set for all modules connected to the CS31 bus.

Caution: There is no fix connection between module address and the input/output addresses of the channels. The input/output addresses are assigned automatically and change when inserting new modules.

When setting the module address, observe the following rules:

1. It is recommended to set a unique address for each module.
2. It is allowed to specify the same module address for a digital module and an analog module, but this is not recommended.
3. If the same module address is set for a digital input module and a digital output module, only the first module in the PLC configuration is detected. This also applies to analog modules!
4. DIL switch 8 (used to allocate the channels 8..15 for series 90) is ignored as all inputs/outputs are byte-oriented.
5. In case of expandable CS31 modules, the maximum configurations have to be observed.

Input/output modules connected to COM1 occupy the following I/O area:

COM1: %IB1000 .. %IB1999 or %QB1000 .. %QB1999

Further parameters and settings for the CS31 modules are optional and can be found in the descriptions for the individual modules.

The S500 modules are described here: S500 module description. AC31 modules are described in the 907 AC 1131 documentation.

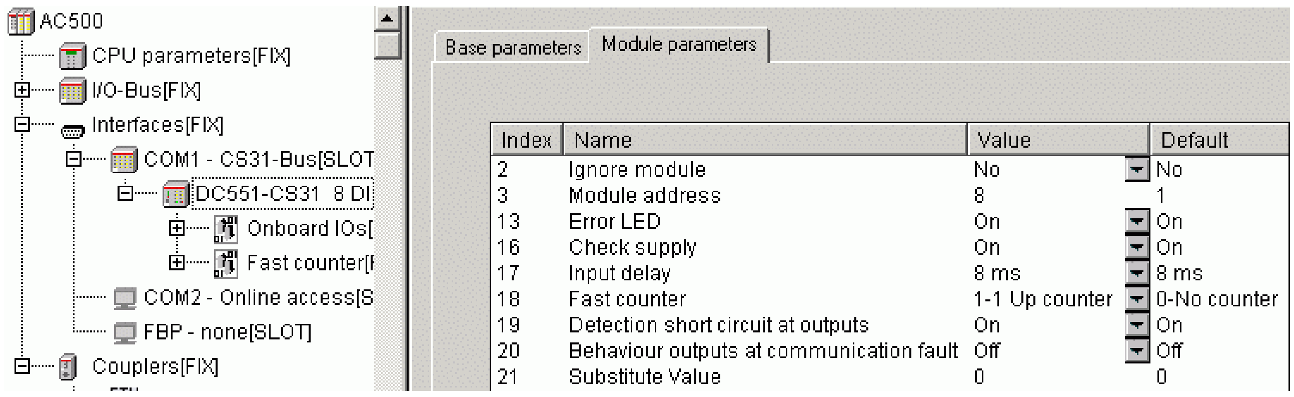
Connecting the DC551 and S500 I/O devices to the CS31 bus

The base module "DC551-CS31" is available in two versions in the PLC configuration:

1. DC551-CS31 8 DI + 16 DC / without fast counter
The addresses 00...69 can be set at the module and in the PLC configuration.

2. DC551-CS31 8 DI + 16 DC + 2FC / with 2 fast counters
The hardware addresses 70...99 can be set at the module. This corresponds to the module addresses 00...29 with activated counter. In the PLC configuration and at the block CNT_DC551, the module address (00..29) is set.
(See also library Counter_AC500_V11.LIB / CNT_DC551).

The following parameters can be set in the PLC configuration for the DC551:



Index	Name	Value	Default
2	Ignore module	No	No
3	Module address	8	1
13	Error LED	On	On
16	Check supply	On	On
17	Input delay	8 ms	8 ms
18	Fast counter	1-1 Up counter	0-No counter
19	Detection short circuit at outputs	On	On
20	Behaviour outputs at communication fault	Off	Off
21	Substitute Value	0	0

Parameter	Default value	Value	Meaning
Ignore module **see remark 1**	No	No	It is checked whether the module exists on the CS31 bus.
		Yes	Module is not checked. -> available as of CPU firmware V1.2.0 and PS501 V1.2
Module address	0	0...69	Module address of the DC551 without fast counter
		0...29	Module address of the DC551 with fast counter
Error-LED **see remark 2**	On	On	The error LED lights up for errors of all classes, no failsafe function activated.
		Off_by_E4	Warnings (E4) are not indicated by the error LED, no failsafe function activated.
		Off_by_E3	Warnings (E4) and light errors (E3) are not indicated by the error LED. No failsafe function activated.
		On+failsafe	The error LED lights up for errors of all classes and the failsafe function of the CS31 bus is activated. -> available as of CPU firmware V1.2.0, DC551 firmware V1.9 and PS501 V1.2
		Off_by_E4+failsafe	Warnings (E4) are not indicated by the error LED, the failsafe function of the CS31 bus is activated. -> available as of CPU firmware V1.2.0, DC551 firmware V1.9 and PS501 V1.2
		Off_by_E3+failsafe	Warnings (E4) and light errors (E3) are not indicated by the error LED, the failsafe function of the CS31 bus is activated. -> available as of CPU firmware V1.2.0, DC551 firmware V1.9 and PS501 V1.2
Check supply	On	On	Control voltage monitoring ON
		Off	Control voltage monitoring OFF
Input delay	8 ms	0.1 / 1 / 8 / 32 ms	Input delay 0.1 / 1 / 8 / 32 ms
Fast counter	0-No counter	0-No counter	Operation mode of the fast counter (see also hardware / description of fast counters)
Detection short-circuits at outputs	On	On	Output short-circuit detection ON
		Off	Output short-circuit detection OFF
Behaviour of outputs at communication fault **see remark 2**	Off	Off	Behavior of outputs at communication faults on the CS31 bus OFF
		Last value	Last value -> available as of CPU firmware V1.2.0, DC551 firmware V1.9 and PS501 V1.2
		Substitute value	Substitute value -> available as of CPU firmware V1.2.0, DC551 firmware V1.9 and PS501 V1.2
		Last value 5 sec.	Last value for 5 seconds -> available as of CPU firmware V1.2.0, DC551 firmware V1.9 and PS501 V1.2
		Substitute value 5 sec.	Substitute value for 5 seconds -> available as of CPU firmware V1.2.0, DC551 firmware V1.9 and PS501 V1.2

		Last value 10 sec.	Last value for 10 seconds -> available as of CPU firmware V1.2.0, DC551 firmware V1.9 and PS501 V1.2
		Substitute value 10 sec.	Substitute value for 10 seconds -> available as of CPU firmware V1.2.0, DC551 firmware V1.9 and PS501 V1.2
Substitute value **see remark 2**	0	0...65535 0000_{hex}...$FFFF_{hex}$	Substitute value for the outputs, one bit per output, bit 0=C8 .. bit 15=C23 -> available as of CPU firmware V1.2.0, DC551 firmware V1.9 and PS501 V1.2

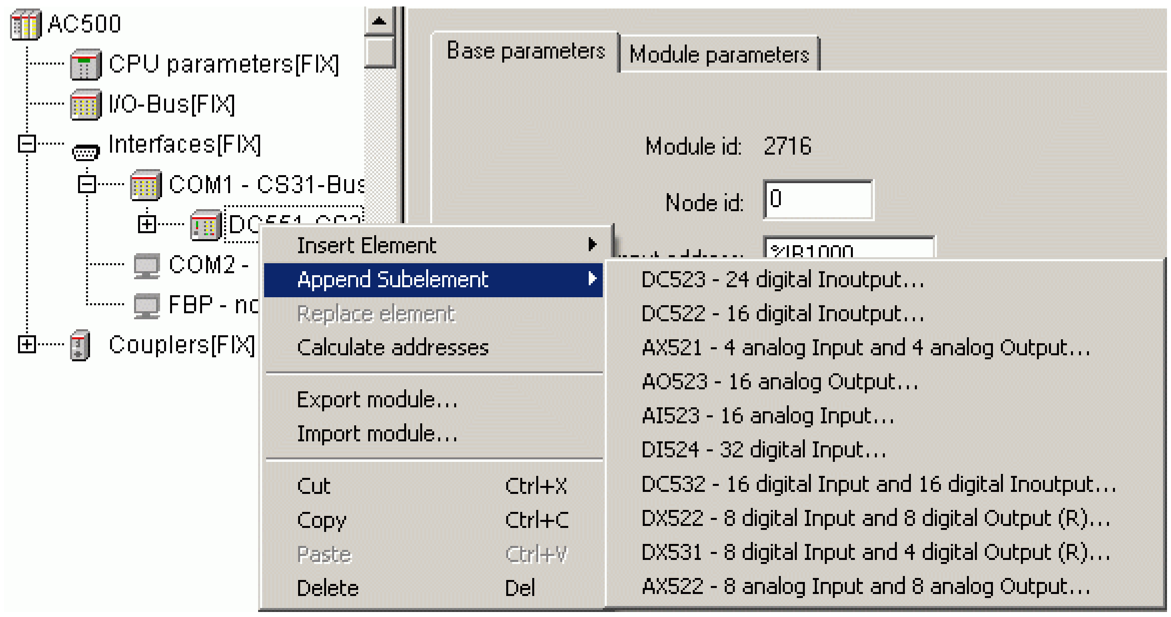
Remark 1: Ignore module

A detailed description of the parameter "Ignore module" can be found in the chapter "The module parameter 'Ignore module' of S500 I/O devices".

Remark 2: Failsafe function of CS31 bus

Further information on the failsafe function of the CS31 bus are contained in the chapter "The failsafe function of S500 I/O devices".

Further S500 modules can be coupled to the base module "DC551-CS31" via the I/O bus. Right-clicking the element "DC551-CS31" in the configuration tree and selecting the menu item "Append Subelement" displays all available input/output modules that can be added to the module "DC551-CS31".


AC500
CPU parameters[FIX]
I/O-Bus[FIX]
Interfaces[FIX]
COM1 - CS31-Bus
COM2 -
FBP - nc
Couplers[FIX]
Base parameters
Module parameters
Insert Element
Append Subelement
Replace element
Calculate addresses
Export module...
Import module...
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Delete Del
DC523 - 24 digital Inoutput...
DC522 - 16 digital Inoutput...
AX521 - 4 analog Input and 4 analog Output...
AO523 - 16 analog Output...
AI523 - 16 analog Input...
DI524 - 32 digital Input...
DC532 - 16 digital Input and 16 digital Inoutput...
DX522 - 8 digital Input and 8 digital Output (R)...
DX531 - 8 digital Input and 4 digital Output (R)...
AX522 - 8 analog Input and 8 analog Output...

Module id: 2716

Node id: 0

%IB1000

A maximum of 7 expansions with a total of 240DI/240DO and 32AI/32AO can be appended to the module.

The following **peculiarities concerning the CS31 bus in the AC500** must be observed when addressing S500 I/O devices at the CS31 bus:

1. A CS31 software module can occupy a maximum of
   -> 15 bytes of inputs and 15 bytes of outputs in the digital area.
   This corresponds to 15 x 8 = 120 digital inputs and 120 outputs.
2. A CS31 software module can allocate a maximum of
   -> 8 words of inputs and 8 words of outputs in the analog area.
3. A maximum of 31 of these CS software modules are allowed for connection to the CS31 bus.
4. If a device has more than 15 bytes or 8 words of inputs or outputs, it occupies 2 or more of the 31 CS31 software modules.

5. The DC551 can internally manage **2 CS31 software modules in the digital area and 4 CS31 software modules in the analog area**. This corresponds to a maximum of:
   - 240 digital inputs (2 x 15 bytes) and
   - 240 digital outputs (2 x 15 bytes) and
   - 32 analog inputs (4 x 8 words) and
   - 32 analog outputs (4 x 8 words).

6. Address setting is done at the DC551 using two rotary switches at the module&rsquo;s front plate.

7. To enable the fast counter of the DC551, the hardware address (HW_ADR) has to be set to the module address + 70. With activated fast counter, the module addresses 0..28 (hardware address setting 70..98) are allowed.
   Then, the DC551 registers as 2 CS31 software modules using the module address (hardware address 70), once in the digital area and once in the analog area.

8. CS31 software module 1 in digital area:
   -> registers using the module address.
   CS31 software module 2 in digital area:
   -> registers using module address+7 and bit "Channel >= 7" set.
   CS31 software module 1 in analog area:
   -> registers using the module address.
   CS31 software module 2 in analog area:
   -> registers using module address and bit "Channel >= 7" set.
   CS31 software module 3 in analog area:
   -> registers using the module address+1.
   CS31 software module 4 in analog area:
   -> registers using module address+1 and bit "Channel >= 7" set.

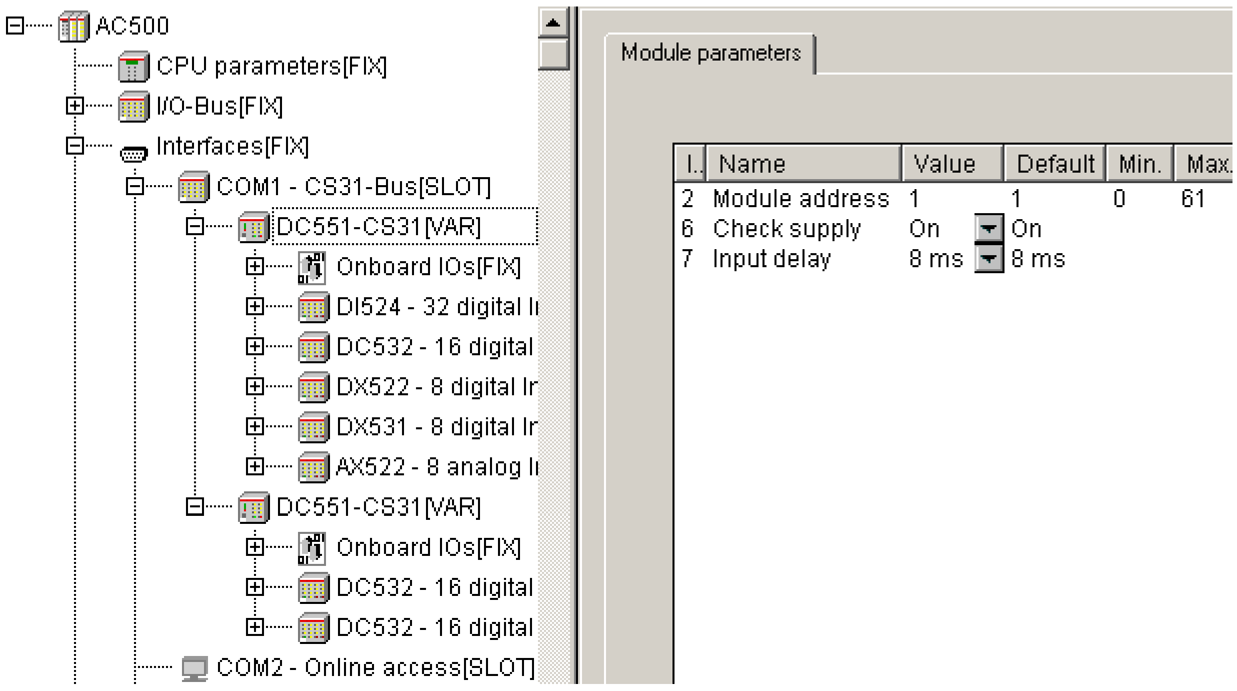
9. The DC551 can manage a maximum of 255 parameters.
   This does not cause any restrictions in all configurations with the currently available S500 I/O devices.

10. The next free address for a DC551 is derived from the highest address occupied in the digital area or the analog area of the previous DC551.

11. When connecting several S500 expansion modules to a DC551 via the I/O bus, their inputs and outputs follow the DC551's inputs and outputs without gap. Such a cluster can occupy up to 5 CS31 software modules.

12. A maximum of 7 S500 expansion modules (extensions) can be connected to a DC551.

A configuration consisting of two combined input/output modules could look as follows:


AC500
CPU parameters[FIX]
I/O-Bus[FIX]
Interfaces[FIX]
COM1 - CS31-Bus[SLOT]
DC551-CS31[VAR]
Onboard IOs[FIX]
DI524 - 32 digital I
DC532 - 16 digital
DX522 - 8 digital Ir
DX531 - 8 digital Ir
AX522 - 8 analog I
DC551-CS31[VAR]
Onboard IOs[FIX]
DC532 - 16 digital
DC532 - 16 digital
COM2 - Online access[SLOT]

Module parameters

I.	Name	Value	Default	Min.	Max.
2	Module address	1	1	0	61
6	Check supply	On	On		
7	Input delay	8 ms	8 ms		

Note: The fast counters of the input/output modules (e.g., "DC532") are only available if the modules are connected to the CPU's I/O bus.

Summary of input/output data of S500 I/O devices

The following input/output data and parameters are available with S500 I/O devices:

Device	ID	I/O range	Digital area		Analog area		Para-meter
			Inputs	Outputs	Inputs	Outputs	
			Byte	Byte	Words	Words	Byte
DC551	2716	8 DI + 16 DC	3	2	0	0	15
DC551+FC	2715	8 DI + 16 DC + FC	5	4	4	8	16
AI523	1515	16 AI	0	0	16	0	36
AO523	1510	16 AO	0	0	0	16	41
AX521	1505	4 AI + 4 AO	0	0	4	4	23
AX522	1500	8 AI + 8 AO	0	0	8	8	39
DC522	1220	16 DC	2	2	0	0	8
DC523	1215	24 DC	3	3	0	0	10
DC532	1200	16 DI + 16 DC	4	2	0	0	8
DI524	1000	32 DI	4	0	0	0	4
DX522	1210	8 DI + 8 DX	1	1	0	0	6
DX531	1205	8 DI + 4 DX	1	1	0	0	6

Examples of impossible configurations

Due to the peculiarities concerning the CS31 bus and the DC551 described at the beginning of this chapter, some configurations cannot be realized. Here are some examples:

Example 1: DC551 + 6 x DC532

Device	I/O range	Digital area		Analog area		Para-meter
		Inputs	Outputs	Inputs	Outputs	
		Byte	Byte	Words	Words	Byte
DC551	8 DI + 16 DC	3	2	0	0	15
DC532	16 DI + 16 DC	4	2	0	0	9
DC532	16 DI + 16 DC	4	2	0	0	9
DC532	16 DI + 16 DC	4	2	0	0	9
DC532	16 DI + 16 DC	4	2	0	0	9
DC532	16 DI + 16 DC	4	2	0	0	9
DC532	16 DI + 16 DC	4	2	0	0	9
DC532	16 DI + 16 DC	4	2	0	0	9
Total	**120 DI + 128 DC**	**31**	**16**	**0**	**0**	**78**

This configuration is not possible because the DC551 can manage a maximum of 30 bytes in the digital area (= 120 inputs/outputs).

Example 2: DC551 + 5 (or more) AX522

Device	I/O range	Digital area		Analog area		Para-meter
		Inputs	Outputs	Inputs	Outputs	
		Byte	Byte	Words	Words	Byte
DC551	8 DI + 16 DC	3	2	0	0	15
AX522	8 AI + 8 AO	0	0	8	8	40
AX522	8 AI + 8 AO	0	0	8	8	40
AX522	8 AI + 8 AO	0	0	8	8	40
AX522	8 AI + 8 AO	0	0	8	8	40
AX522	8 AI + 8 AO	0	0	8	8	40
Total	**8 DI + 16 DC + 32 AI + 32 AO**	**3**	**2**	**40**	**40**	**215**

This configuration is not possible because the DC551 can manage a maximum of 32 words in the analog area. For 6 or 7 AX522, the number of analog channels increases accordingly.

Example 3: DC551 + 3 (or more) AO523

Device	I/O range	Digital area		Analog area		Para-meter
		Inputs	Outputs	Inputs	Outputs	
		Byte	Byte	Words	Words	Byte
DC551	8 DI + 16 DC	3	2	0	0	15
AO523	16 AO	0	0	0	16	42
AO523	16 AO	0	0	0	16	42
AO523	16 AO	0	0	0	16	42
Total	**8 DI + 16 DC + 48 AO**	**3**	**2**	**0**	**48**	**141**

This configuration is not possible because the DC551 can manage a maximum of 32 words in the analog area. For each further AO523, the number of analog channels increases accordingly.

Example 4: DC551 + 3 (or more) AI523

Device	I/O range	Digital area		Analog area		Para-meter
		Inputs	Outputs	Inputs	Outputs	
		Byte	Byte	Words	Words	Byte
DC551	8 DI + 16 DC	3	2	0	0	15
AI523	16 AI	0	0	16	0	37
AI523	16 AI	0	0	16	0	37
AI523	16 AI	0	0	16	0	37
Total	**8 DI + 16 DC + 48 AI**	**3**	**2**	**48**	**0**	**126**

This configuration is not possible because the DC551 can manage a maximum of 32 words in the analog area. For each further AI523, the number of analog channels increases accordingly.

Example 5: DC551 + 5 (or more) PD501

Device	I/O range	Digital area		Analog area		Para-meter
		Inputs	Outputs	Inputs	Outputs	
		Byte	Byte	Words	Words	Byte
DC551	8 DI + 16 DC	3	2	0	0	15
PD501	8 DO + 8 AI	0	1	8	0	21
PD501	8 DO + 8 AI	0	1	8	0	21
PD501	8 DO + 8 AI	0	1	8	0	21
PD501	8 DO + 8 AI	0	1	8	0	21
PD501	8 DO + 8 AI	0	1	8	0	21
Total	**8 DI + 16 DC + 48 DO + 40 AI**	**3**	**7**	**40**	**0**	**120**

This configuration is not possible because the DC551 can manage a maximum of 32 words in the analog area. For 6 or 7 PD501, the number of analog channels increases accordingly.

Example 6: DC551 with FC + 2 (or more) AO523

Device	I/O range	Digital area		Analog area		Para-meter
		Inputs	Outputs	Inputs	Outputs	
		Byte	Byte	Words	Words	Byte
DC551	8 DI + 16 DC + FC	5	4	4	8	16
AO523	16 AO	0	0	0	16	42
AO523	16 AO	0	0	0	16	42
Total	**8 DI + 16 DC + FC + 32 AO**	**5**	**4**	**4**	**40**	**100**

This configuration is not possible because the DC551 can manage a maximum of 32 words in the analog area. For each further AO523, the number of analog channels increases accordingly.

Example 7: DC551 with FC + 2 (or more) AI523

Device	I/O range	Digital area		Analog area		Para-meter
		Inputs	Outputs	Inputs	Outputs	
		Byte	Byte	Words	Words	Byte
DC551	8 DI + 16 DC + FC	5	4	4	8	16
AI523	16 AI	0	0	16	0	37
AI523	16 AI	0	0	16	0	37
Total	**8 DI + 16 DC + FC + 32 AI**	**5**	**4**	**36**	**8**	**90**

This configuration is not possible because the DC551 can manage a maximum of 32 words in the analog area. For each further AI523, the number of analog channels increases accordingly.

3.4.6 The setting 'COMx - SysLibCom'

As of Control Builder version V1.2 and firmware version V1.2.0 the protocol

- 'COMx - SysLibCom'

is available.

If the protocol 'COMx - SysLibCom' is selected for the serial interface COMx, the interface is prepared for operation with the blocks contained in the library "SysLibCom.lib" and the according protocols.

The library SysLibCom.lib contains the following functions:

Function	Meaning	Note
SysComClose	Closes an interface	
SysComGetVersion2300	Internal version synchronization	Only internal
SysComOpen	Opens an interface	
SysComRead	Reads data from an interface	
SysComSetSettings	Parameterization of an interface	
SysComSetSettingsEx	Extended parameterization of an interface	Currently not supported
SysComWrite	Write data to an interface	

Note: All blocks in the "SysLibCom.lib" library are functions. The blocks are processed until the according action is completed or processing is aborted due to a possibly set timeout. During sending, the block waits, for example, until the characters have been actually output.
This can cause the suspension of the task in case of too small task cycle times!

The interface is initialized in the PLC configuration according to the defined settings when starting the PLC, after a download or after a reset. The following settings are possible:

Parameter	Default value	Value	Meaning
Enable login **see remark 1**	Disabled	Disabled	There is no check with regard to the Control Builder login telegram.
		Enabled	Telegrams received are checked with regard to the Control Builder login sequence. If the sequence is detected, the protocol setting is changed to 'Online access'.
RTS control **see remark 2**	None	None	No RTS control (direction control)
		telegram	RTS control activated (absolutely necessary for RS 485!)

TLS **see remark 2**	0	0...65535	Carrier lead time in [ms] (TLS > CDLY)
CDLY see remark 2	0	0...65535	Carrier delay time in [ms] (CDLY <= TLS)
Character timeout **see remark 3**	0	0...65535	Character timeout in characters (must be 0 if Telegram ending selection = Character timeout)
Telegram ending selection **see remark 3**	None	**None**	No telegram ending identifier
		String (check receive)	2 characters, e.g. <CR><LF> (16#0d, 16#0a -> 16#0d0a) in parameter "Telegram ending value" **Setting not recommended!**
		Telegram length	Telegram ending identifier set by telegram length **Setting not recommended!**
		Duration	Telegram ending identifier set by time **Setting not recommended!**
		Character timeout	Telegram ending identifier set by character timeout
Telegram ending character **see remark 3**	0	0...1	Number of end characters in case of telegram ending selection "String"
Telegram ending value **see remark 3**	0	0...65535	Telegram ending identifier value for settings "Duration", "Character timeout" and "String"
Handshake	None	None	No handshake
		RTS/CTS	Hardware handshake
		XON/XOFF	Not yet implemented
		3964R master	Not yet implemented
		3964R slave	Not yet implemented
Baudrate	19200	300 1200 4800 9600 19200 38400 57600 115200 125000 187500	Character length in bits/s
Parity	None	None	No parity check
		Odd	Odd parity
		Even	Even parity
		Mark	Parity bit := TRUE
		Space	Parity bit := FALSE
Data bits	8	5, 6, 7, 8	Character length in bits/character
Stop bits	1	1, 2	Number of stop bits

Remark 1: Enable login

See remark 1 under "ASCII" protocol settings

Remark 2: Usage of modems

See remark 2 under "ASCII" protocol settings

Remark 3: Telegram ending identifier

See remark 3 under "ASCII" protocol settings

Because the receive function SysComRead() interrupts the processing of the user program until the ending criteria (telegram ending selection or timeout) is detected, it is recommended to set the telegram ending selection only to the following values:

- None
- Character timeout

During processing of the user program, the following parameters can be changed using the function SysComSetSettings():

- Baudrate
- Number of stop bits
- Parity

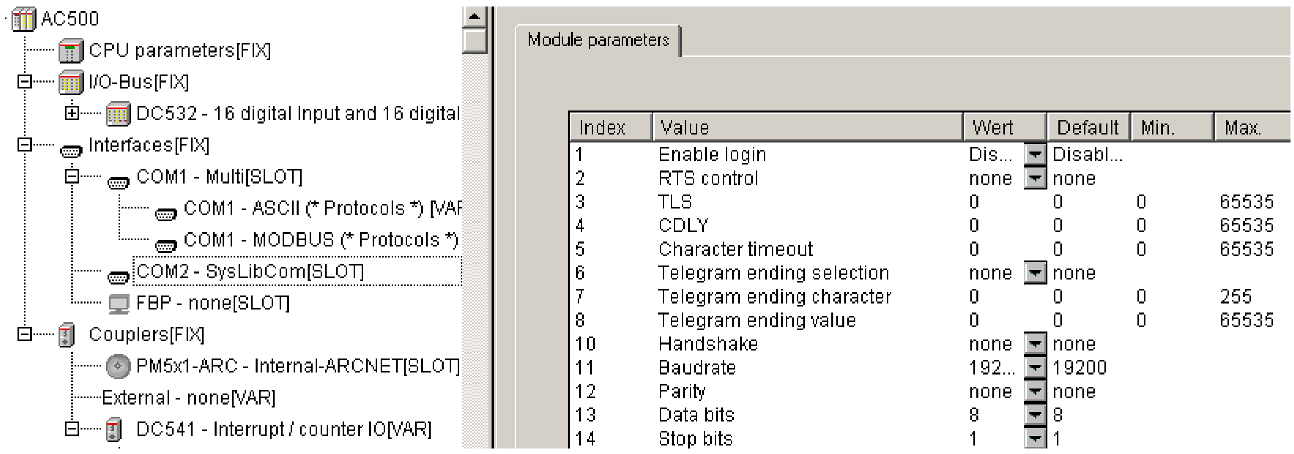
This is done by adding the parameters to the structure COMSETTINGS. The structure is as follows:

Parameter	**Type**	**Default value**	**Meaning / valid values**
dwBaudRate	DWORD	none	Baudrate 300, 1200, 4800, 9600, 14400, 19200, 38400 57600, 115200, 125000, 187500 **Warning:** The structure must have a valid value assigned to it. The function reports an error for any invalid values (also 0).
byStopBits	BYTE	0	Number of stop bits 0=1, 1=1,5, 2=2 stop bits
byParity	BYTE	0	Parity 0=No, 1=odd, 2=even
dwTimeout	DWORD	0	Currently not supported! Specified values are ignored.
dwBufferSize	DWORD	0	
dwScan	DWORD	0	

Example for sending/receiving with "SysLibCom"

The following example shows how data are sent/received with the protocol "SysLibCom".
-> Telegrams of 32 bytes length are to be received and sent.

1. Setting in PLC configuration:



AC500
CPU parameters[FIX]
I/O-Bus[FIX]
DC532 - 16 digital Input and 16 digital
Interfaces[FIX]
COM1 - Multi[SLOT]
COM1 - ASCII (* Protocols *) [VAF
COM1 - MODBUS (* Protocols *)
COM2 - SysLibCom[SLOT]
FBP - none[SLOT]
Couplers[FIX]
PM5x1-ARC - Internal-ARCNET[SLOT]
External - none[VAR]
DC541 - Interrupt / counter IO[VAR]
Module parameters

Index	Value	Wert	Default	Min.	Max.
1	Enable login	Dis...	Disabl...		
2	RTS control	none	none		
3	TLS	0	0	0	65535
4	CDLY	0	0	0	65535
5	Character timeout	0	0	0	65535
6	Telegram ending selection	none	none		
7	Telegram ending character	0	0	0	255
8	Telegram ending value	0	0	0	65535
10	Handshake	none	none		
11	Baudrate	192...	19200		
12	Parity	none	none		
13	Data bits	8	8		
14	Stop bits	1	1		

2. Declaration part of the program PROGRAM proSysLibCom_Test

```
VAR
    strComSettings    : COMSETTINGS;          (* Structure of COM settings *)
    dwHandle          : DWORD;
    byStep            : BYTE;                 (* Step chain *)
    dwRead            : DWORD;                (* Number of characters received *)
    dwWritten         : DWORD;                (* Number of characters sent *)
    bEnSend           : BOOL;                 (* Enable sending *)
    byCom             : BYTE := COM2;         (* COM number *)
    dwBaudrate        : DWORD := 19200;       (* Baudrate *)
    wLenRec           : WORD := 32;           (* Number of characters to be received *)
    wLenTele          : WORD := 32;           (* Telegram length, here 32 characters for
                                              example *)
    wLenSend          : WORD := 32;           (* Number of characters to be sent, for example
                                              32 characters *)
    dwTimeoutSend     : DWORD := 0;           (* Timeout in [ms] for sending *)
    dwTimeoutRec      : DWORD := 0;           (* Timeout in [ms] for receiving *)
    abyRecBuffer      : ARRAY[0..271] OF      (* Receive butter, 272 bytes min.! *)
                      BYTE;
    abyTeleBuffer     : ARRAY[0..543] OF      (* Telegram buffer, 2 x receive buffer min. *)
                      BYTE;
    aby SendBuffer    : ARRAY[0..271] OF      (* Send buffer, telegram length max.! *)
                      BYTE;
    strDataRec        : StrucReceiveData;     (* Structure of receive telegram *)
    strDataSend       : StrucSendData         (* Structure of send telegram *)
END_VAR
```

3. Code part of the program

-> Processing of a step chain containing the following steps

```
CASE byStep OF
0:      (* Step 0: Open the interface COMx -> SysComOpen -> get handle *)
        strComSettings.Port := byCom;                    (* COM_Number *)
        dwHandle := SysComOpen(strComSettings.Port);     (* Open COM interface -> get
                                                         handle *)
        byStep := SEL( dwHandle <> INVALID_HANDLE, 250,  (* handle ok -> Step 1,
        1);                                              otherwise error step 250 *)

1:      (* Step 1: Transfer of COMx interface parameters *)
        strComSettings.dwBaudRate := dwBaudrate;         (* Set baudrate *)
        (* Enter at this point the number of stop bits and parity, if necessary *)

        (* set COM settings -> if OK, run step 2, in case of an error step 250 *)
        byStep := SEL( SysComSetSettings(dwHandle, ADR(strComSettings)), 250, 2);

2:      (* Step 2: Initialization completed successfully -> now receiving and/or sending *)

        (* Receive data:
        read all data received since last run, but wLenRec max.! *)
        dwRead := SysComRead( dwHandle,
                    ADR(abyRead),
                    WORD_TO_DWORD(wLenRec),
                    dwTimeoutRec);                       (* READ DATA *)
        IF (dwRead > 0) THEN (* Number of characters received; in bytes *)
            (* here, ignore for example all characters until valid telegram start detected *)

            (* Number of receiving cycles for the telegram *)
            dwNumReadPerTele[byCom] := dwNumReadPerTele[byCom] + 1;
            (* Copy data to buffer *)
            SysMemCpy(   dwDest := ADR(abySumDataRead[dwSum]DataRead]),
                         dwSrc := ADR(abyRead[0]),
                         dwCount := dwRead );
            (* Sum of read data of a telegram *)
            dwSumDataRead := dwSumDataRead + dwRead;
            IF dwSumDataRead >= wLenTele THEN            (* Telegram complete ? *)
                    dwRecCount := dwRecCount +1;         (* Number of telegrams received *)
                    (* Copy received telegram to structure strDataRec *)
                    SysMemCpy( dwDest := ADR(strDataRec,
                         dw Src := ADR(abySumDataRead[0]),
                         dwCount := wLenTele );
                      dwNumReadPerTele := 0;             (* init for following telegram *)

                    dwSumDataRead := 0;

                    (* here the evaluation of data starts !!! *)

            END_IF;                                      (* Telegram complete *)
```

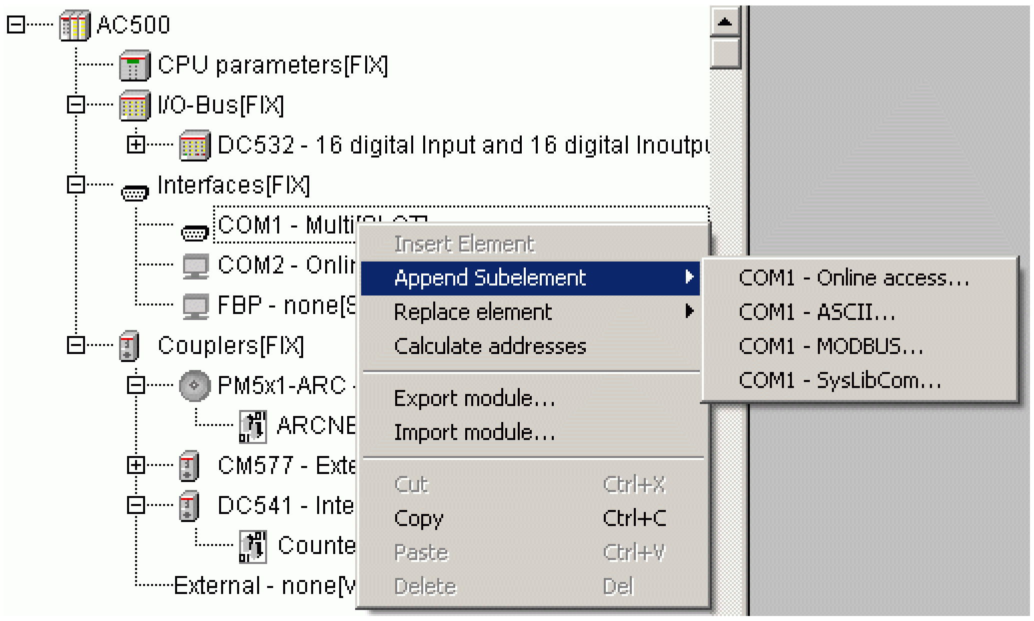
```
          END_IF;                                                    (* Data received *)

          (* Send data *)
          (* Enabling the sending of data can be done, for example, cyclical or by program control *)
          IF bEnSend THEN                                            (* Enable sending *)
               (* Copy data to be sent to send buffer *)
               SysMemCpy(      dwDest: := ADR(abyDataSend[0]),
                               dwSrc := ADR(strDataSend),
                               dwCount := wLenSend);
               (* Send data *)
               dwWritten := SysComWrite( dwHandle,
                               ADR(abyDataSend[0]),
                               WORD_TO_DWORD(wLenSend),
                               dwTimeoutSend);                       SEND DATA !!! *)
               IF (dwWritten <> wLenSend THEN
                         byStep := 250;                              (* Error when sending *)
               END_IF;                                               (* dwWritten *)
               bEnSend := FALSE;                                     (* Deactivate enable sending *)
          END_IF;                                                    (* bEnSend *)

250:      (* Step 250: Error step -> Close COMx and start with step 0 *)
          bResult := SysComClose(dwHandle);                          (* Close COM interface *)
          dwHandle := 0;
          byStep := 0;

END_CASE;                                                            (* End of step chain *)
```

3.4.7 The setting 'COMx - Multi'

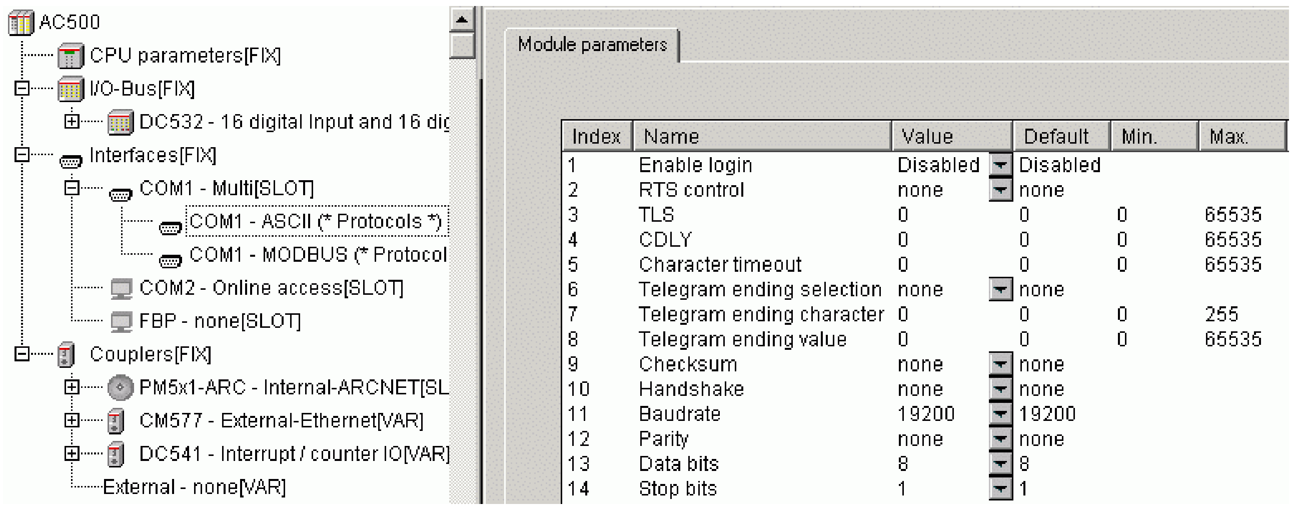
As of Control Builder version V1.2 and firmware version V1.2.0 the protocol 'COMx- Multi' can be used.

If the protocol 'COMx - Multi' is set for the serial interface COMx, the interface is prepared for operation with two selectable protocols.

Both protocols are appended to the protocol 'COMx - Multi' as modules. This is done in the same way than appending input/output modules to the I/O bus:
i.e., by right-clicking the interface, selecting Append Subelement and selecting the desired protocol.



Once both protocols have been appended, the parameters have to be set for each protocol:


AC500
CPU parameters[FIX]
I/O-Bus[FIX]
DC532 - 16 digital Input and 16 dig
Interfaces[FIX]
COM1 - Multi[SLOT]
COM1 - ASCII (* Protocols *)
COM1 - MODBUS (* Protocol
COM2 - Online access[SLOT]
FBP - none[SLOT]
Couplers[FIX]
PM5x1-ARC - Internal-ARCNET[SL
CM577 - External-Ethernet[VAR]
DC541 - Interrupt / counter IO[VAR]
External - none[VAR]
Module parameters

Index	Name	Value	Default	Min.	Max.
1	Enable login	Disabled	Disabled		
2	RTS control	none	none		
3	TLS	0	0	0	65535
4	CDLY	0	0	0	65535
5	Character timeout	0	0	0	65535
6	Telegram ending selection	none	none		
7	Telegram ending character	0	0	0	255
8	Telegram ending value	0	0	0	65535
9	Checksum	none	none		
10	Handshake	none	none		
11	Baudrate	19200	19200		
12	Parity	none	none		
13	Data bits	8	8		
14	Stop bits	1	1		

The protocol parameters are identical to the parameters described for the individual protocols.

When restarting the program, i.e., after switching power ON, a download or a reset, the protocol appended first is always active.

Switching between the protocols is done using the block "COM_SET_PROT" (contained in the library SysInt_AC500_V10.lib). At the block input COM, the number of the serial interface is applied and at the input IDX the protocol index is set. The protocol appended first in the PLC configuration has the index 0, the second protocol the index 1.

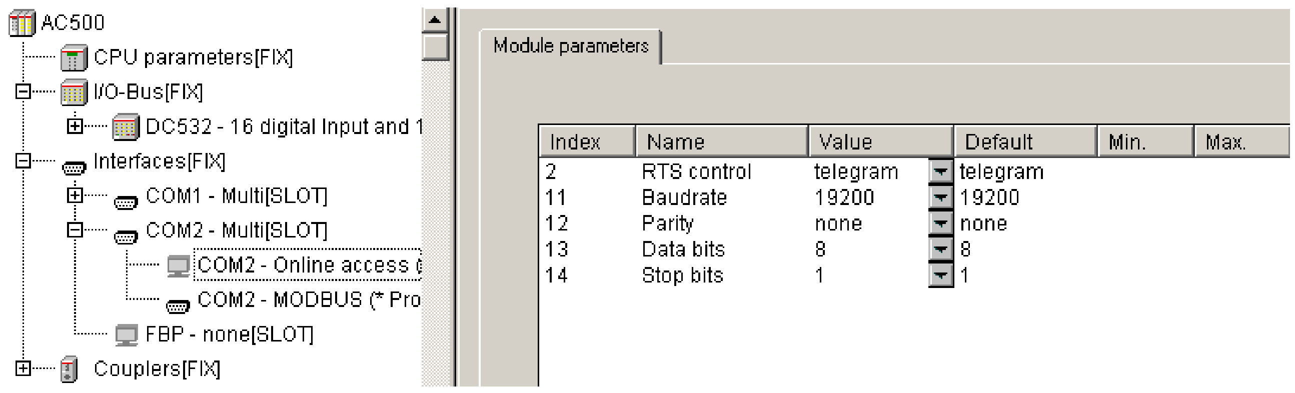
Functions of the block COM_SET_PROT

The block COM_SET_PROT can be used for different functions:

- Switching between two different protocols, for example ASCII / Modbus
- Switching the interface parameters of a protocol, for example changing the baudrate
- Re-initialization of an interface protocol (for example, if an interface "hangs up")
- Switching between "Online access" and ASCII/Modbus/SysLibCom depending on the current PLC mode, for example STOP=Online access, RUN=Modbus (or ASCII, SysLibCom). In this case, the parameter "Enable login" does not have to be activated and the interface can use other interface parameters than required for "Online access" (see the following program example).

Example for switching COM2 between "Online access" and Modbus (master)

1. Setting in the PLC configuration:
-> Protocol with index 0: Online access

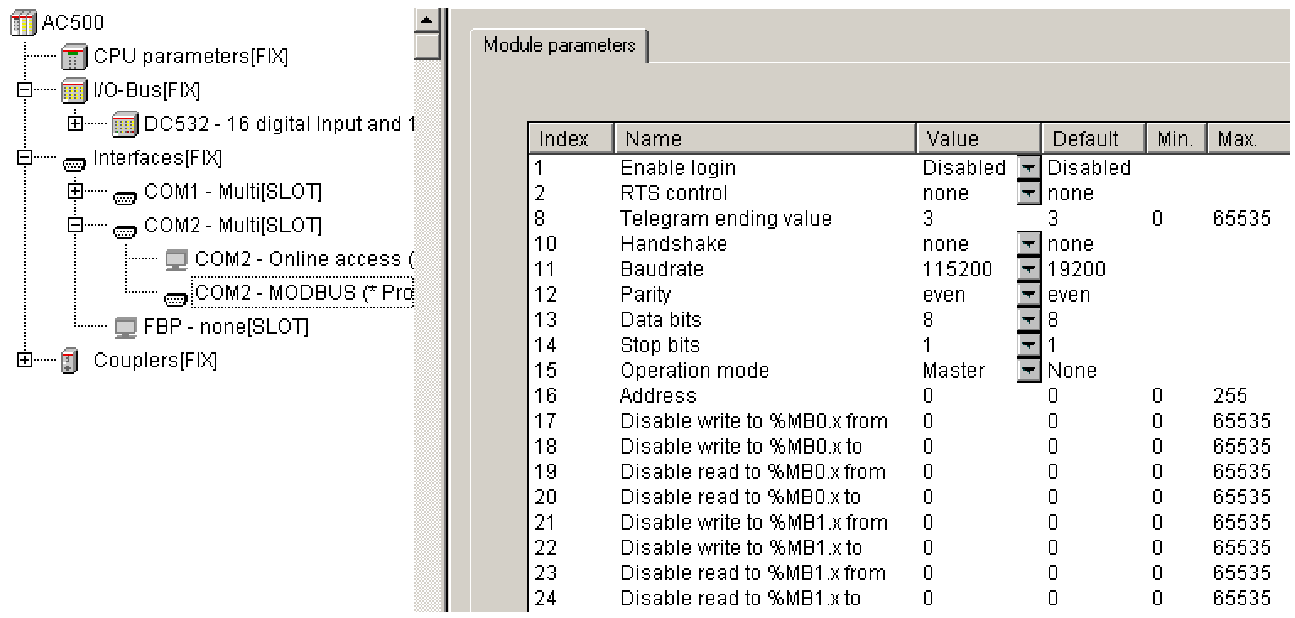


```
AC500
  CPU parameters[FIX]
  I/O-Bus[FIX]
    DC532 - 16 digital Input and 1
  Interfaces[FIX]
    COM1 - Multi[SLOT]
    COM2 - Multi[SLOT]
      COM2 - Online access (
      COM2 - MODBUS (* Pro
    FBP - none[SLOT]
  Couplers[FIX]
```

Module parameters

Index	Name	Value	Default	Min.	Max.
2	RTS control	telegram	telegram		
11	Baudrate	19200	19200		
12	Parity	none	none		
13	Data bits	8	8		
14	Stop bits	1	1		

-> Protocol with index 1: Modbus (master)

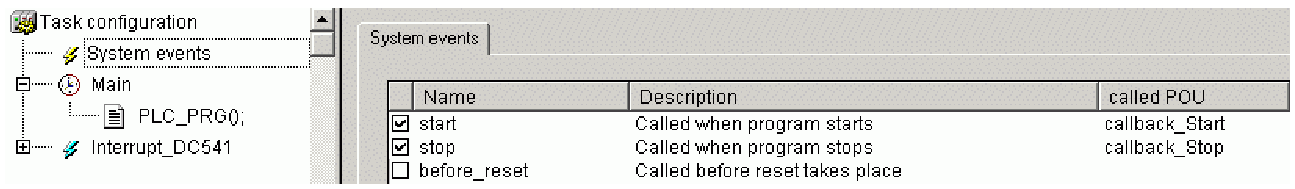


```
AC500
  CPU parameters[FIX]
  I/O-Bus[FIX]
    DC532 - 16 digital Input and 1
  Interfaces[FIX]
    COM1 - Multi[SLOT]
    COM2 - Multi[SLOT]
      COM2 - Online access (
      COM2 - MODBUS (* Pro
    FBP - none[SLOT]
  Couplers[FIX]
```

Module parameters

Index	Name	Value	Default	Min.	Max.
1	Enable login	Disabled	Disabled		
2	RTS control	none	none		
8	Telegram ending value	3	3	0	65535
10	Handshake	none	none		
11	Baudrate	115200	19200		
12	Parity	even	even		
13	Data bits	8	8		
14	Stop bits	1	1		
15	Operation mode	Master	None		
16	Address	0	0	0	255
17	Disable write to %MB0.x from	0	0	0	65535
18	Disable write to %MB0.x to	0	0	0	65535
19	Disable read to %MB0.x from	0	0	0	65535
20	Disable read to %MB0.x to	0	0	0	65535
21	Disable write to %MB1.x from	0	0	0	65535
22	Disable write to %MB1.x to	0	0	0	65535
23	Disable read to %MB1.x from	0	0	0	65535
24	Disable read to %MB1.x to	0	0	0	65535

2. Setting the system events START and STOP in the task configuration:



```
Task configuration
  System events
  Main
    PLC_PRG();
  Interrupt_DC541
```

System events

Name	Description	called POU
☑ start	Called when program starts	callback_Start
☑ stop	Called when program stops	callback_Stop
☐ before_reset	Called before reset takes place	

3. Call of block COM_SET_PROT in system events

```
FUNCTION callback_Start: DWORD
VAR_INPUT
    dwEvent : DINT;
    dwFilter: DINT;
    dwOwner : DINT;
END_VAR
COM_SET_PROT(EN := FALSE); (* for edge creation *)
COM_SET_PROT(EN := TRUE, COM := 2, IDX := 1); (* switch to Modbus *)

FUNCTION callback_Stop : BOOL
VAR_INPUT
    dwEvent : DINT;
    dwFilter: DINT;
    dwOwner : DINT;
END_VAR
COM_SET_PROT(EN := FALSE); (* for edge creation *)
COM_SET_PROT(EN := TRUE, COM := 2, IDX := 0); (* switch to Online access *)
```

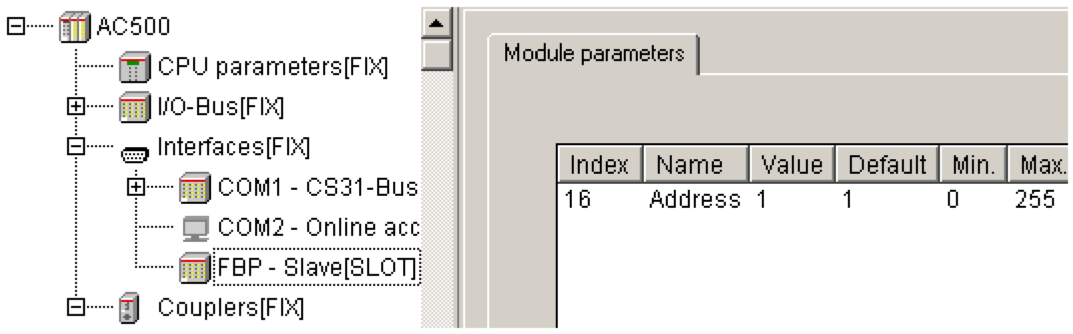
3.5 FBP slave interface configuration (Interfaces / FBP slave)

The FBP slave interface is used to connect the AC500 controllers as fieldbus slave via FieldBusPlug (FBP).

No protocol is set for the FBP interface in the standard configuration (setting "FBP - none").

Note: The FBP interface is configured as "Online access" for PM57x and PM58x (as of firmware version V1.1.7) and PM59x (as of version V1.2.0).
This allows programming via the FBP slave interface using the device UTF21-FBP adapter USB.

Setting the protocol "FBP - Slave" is done using the command "Replace element".



Index	Name	Value	Default	Min.	Max.
16	Address	1	1	0	255

The following setting is possible:

Parameter	Default value	Value	Meaning
Address	0	0...255	Address as FBP slave

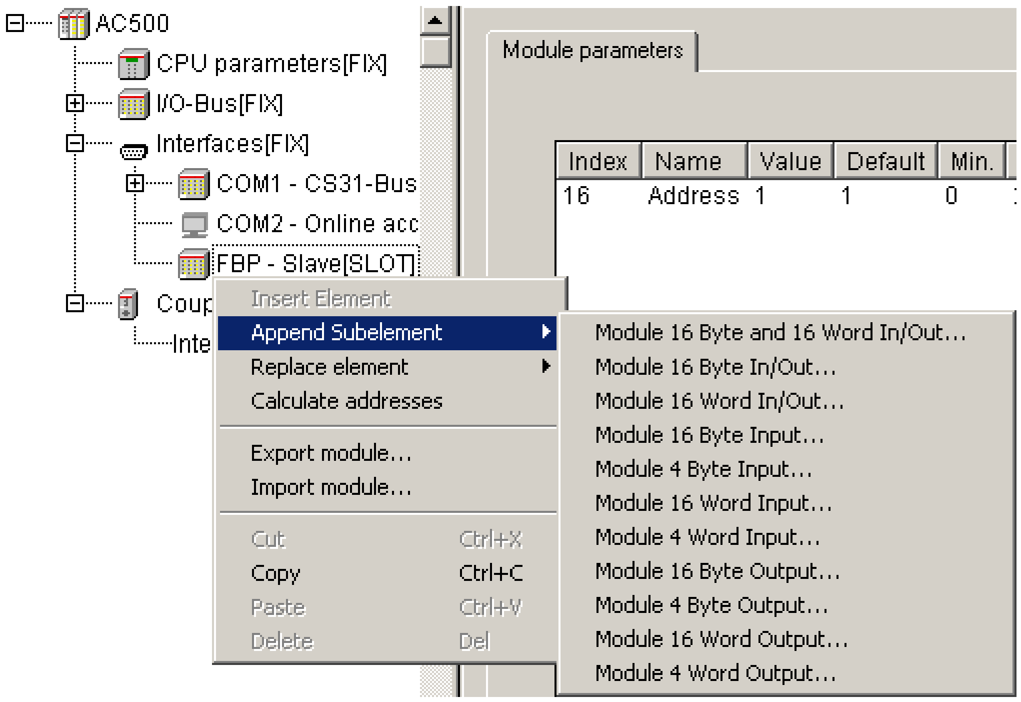
Note: If the FBP slave interface address (ADR > 0) is set using the display/keypad, this address has priority over the PLC configuration setting.

The FBP slave interface occupies the I/O area:

%IB3000 .. %IB3999 or %QB3000 .. %QB3999.

Depending on the fieldbus master, the AC500 CPU can exchange a different amount of input/output data with the master.

Right-clicking the "FBP-Slave" element in the configuration tree opens the context menu where you can change the "FBP" module. Select "Append Subelement". The sub menu displays all available input and output modules for the FBP slave interface:



A maximum of 8 modules can be appended to the FBP slave interface. A module can have a maximum of 16 byte and 16 word inputs and outputs. The number and size of possible modules depends on the used FBP, fieldbus and fieldbus master coupler.

FBP	Fieldbus	I/O range
PDP21	PROFIBUS DP V0	1 module with a maximum of 16 bytes and 16 words (inputs or outputs), for example 1 x "Module 16 Byte and 16 Word In/Out"
PDP22 DP V1 modular	PROFIBUS DP V0/V1	8 modules, but a total of 32 bytes and 128 words in/out, 244 bytes max. per direction, a total of 368 bytes per slave
DNP21	DeviceNet	1 module with a maximum of 16 bytes and 16 words (inputs or outputs), for example 1 x "Module 16 Byte and 16 Word In/Out"
DNP21 modular	DeviceNet	8 modules, but a total of 16 bytes and 16 words (input or output)

The Byte inputs and outputs are provided as BYTE and BOOL and the Word inputs and outputs as WORD, BYTE and BOOL.

The I/O modules saved in the PLC configuration and their addresses must match the entries in the configuration of the respective fieldbus master.

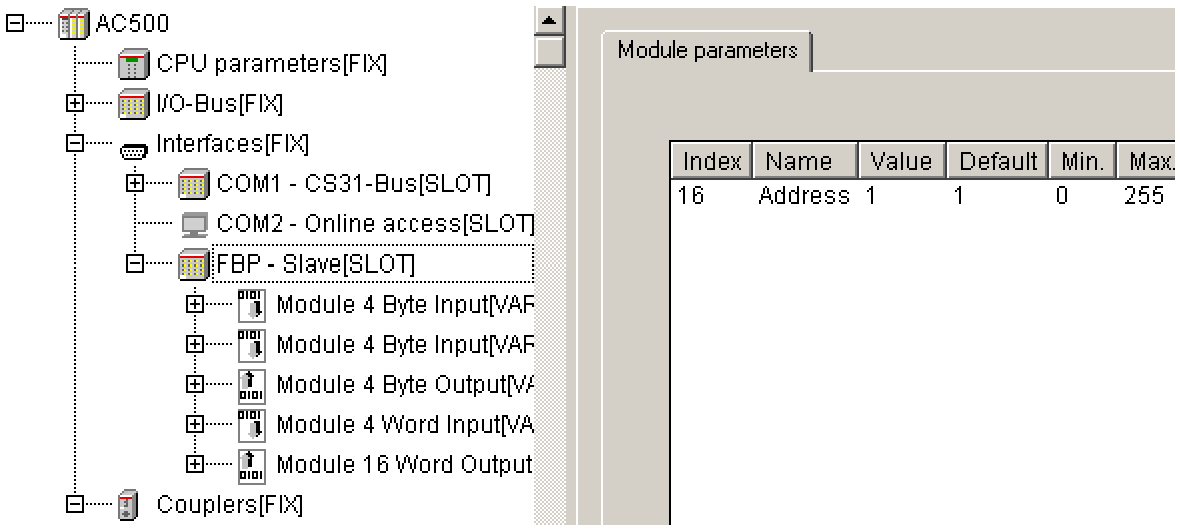
If you want to exchange less data than the maximum allowed amount of I/O data with the fieldbus master, you can setup a configuration consisting of different modules.

In the following example, the AC500 CPU operating as fieldbus slave no. 10 shall exchange 8 Byte inputs, 4 Byte outputs, 4 Word inputs and 16 Word outputs with a fieldbus master.

The following modules are appended using the context menu items:

2 x "4 Byte Input"
1 x "4 Byte Output"
1 x "4 Word Input" and
1 x "16 Word Output".

The final configuration looks as follows:


```
AC500
  CPU parameters[FIX]
  I/O-Bus[FIX]
  Interfaces[FIX]
    COM1 - CS31-Bus[SLOT]
    COM2 - Online access[SLOT]
    FBP - Slave[SLOT]
      Module 4 Byte Input[VAF
      Module 4 Byte Input[VAF
      Module 4 Byte Output[V/
      Module 4 Word Input[VA
      Module 16 Word Output
  Couplers[FIX]
```

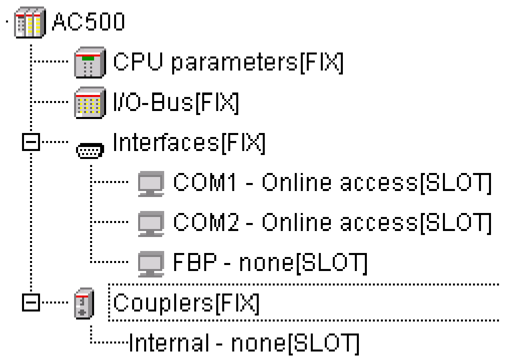
Module parameters

Index	Name	Value	Default	Min.	Max.
16	Address	1	1	0	255

In this configuration, the Byte inputs and outputs are provided as BYTE and BOOL and the Word inputs and outputs as WORD, BYTE and BOOL.

3.6 Coupler configuration (Couplers)

In the standard configuration selected with "File" / "New" or "Extras" / "Standard configuration", the PLC configuration contains no couplers:


```
AC500
  CPU parameters[FIX]
  I/O-Bus[FIX]
  Interfaces[FIX]
    COM1 - Online access[SLOT]
    COM2 - Online access[SLOT]
    FBP - none[SLOT]
  Couplers[FIX]
    Internal - none[SLOT]
```

PLCconf_Coupler1.gif

The following assignment between coupler line and slot applies to the couplers:

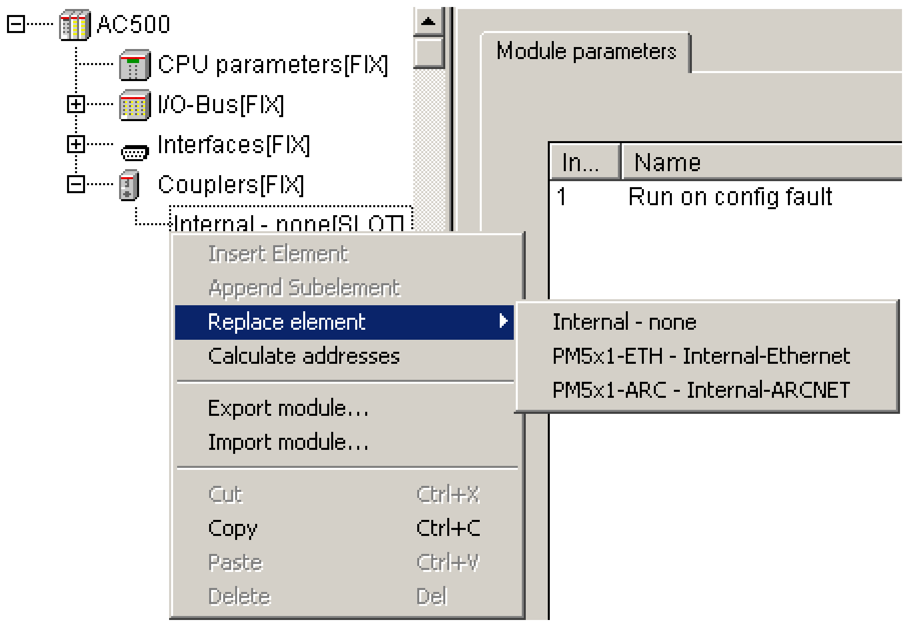
- Line 0 corresponds to the internal coupler (installed in the CPU's housing)
- Line 1 is the coupler installed in the first slot on the left of the CPU
- Line 2 is the coupler installed in the second slot on the left of the CPU
- Line 3 and 4 are installed in the third and fourth slot on the left of the CPU

The allocation of inputs/outputs for the couplers is done slot-oriented and independent of the coupler type.

Note: If a coupler slot is empty, the entry "External none" has to be set for this coupler slot. This is not necessary, if **no** external coupler is inserted.
Example: Internal Ethernet coupler and PROFIBUS coupler in slot 2
1. Replace element: "Internal none" by "PM5x1 Internal Ethernet"
2. Append subelement: "External none" to empty slot 1
3. Append subelement: "CM572 External PROFIBUS DP Master" in slot 2

3.6.1 Configuring the internal coupler

For an AC500 controller with internal coupler, the coupler must be specified in the PLC configuration. This is done by right-clicking the element "Internal - none" in the configuration tree and selecting the sub menu "Replace element". All available internal couplers are shown:

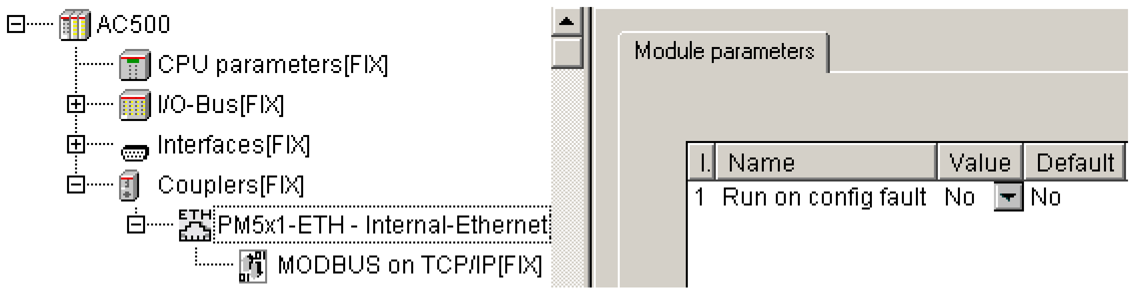


In...	Name
1	Run on config fault

Select the coupler required for the used hardware.

3.6.1.1 The internal Ethernet coupler PM5x1-ETH

The following parameters can be set for the internal Ethernet coupler "PM5x1-ETH - Internal Ethernet":

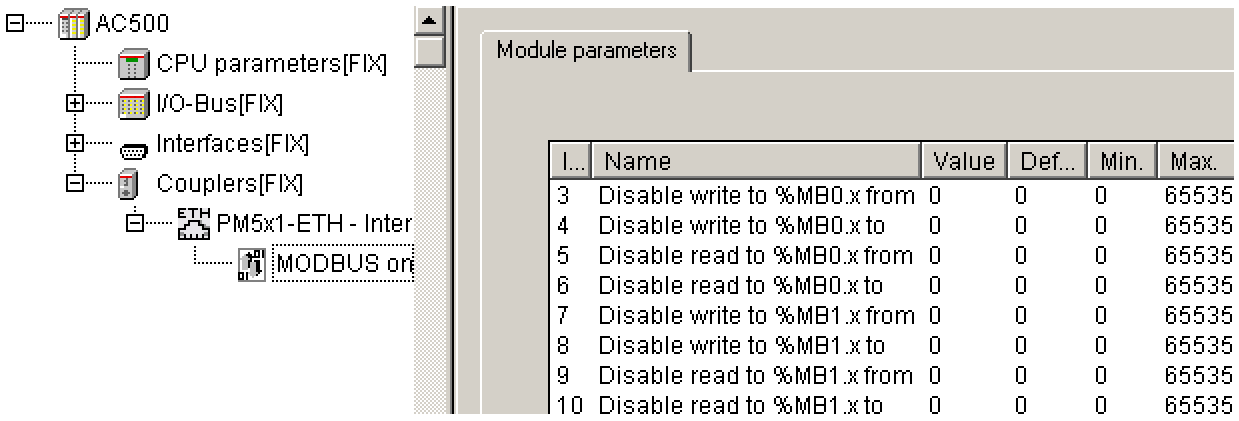


I.	Name	Value	Default
1	Run on config fault	No	No

Parameter	Default value	Value	Meaning
Run on config fault	No	No	In case of a configuration error, the user program is not started.
		Yes	The user program is started even if the internal Ethernet coupler is configured incorrectly.

Only the behavior of the CPU in case of a configuration error of the coupler and the required protocols are set in the PLC configuration. The actual configuration of the Ethernet coupler such as the setting of the IP address is done with the integrated fieldbus configurator SYCON.net (see SYCON.net).

The protocol "MODBUS on TCP/IP" is available by default. Here, the following parameters can be set:

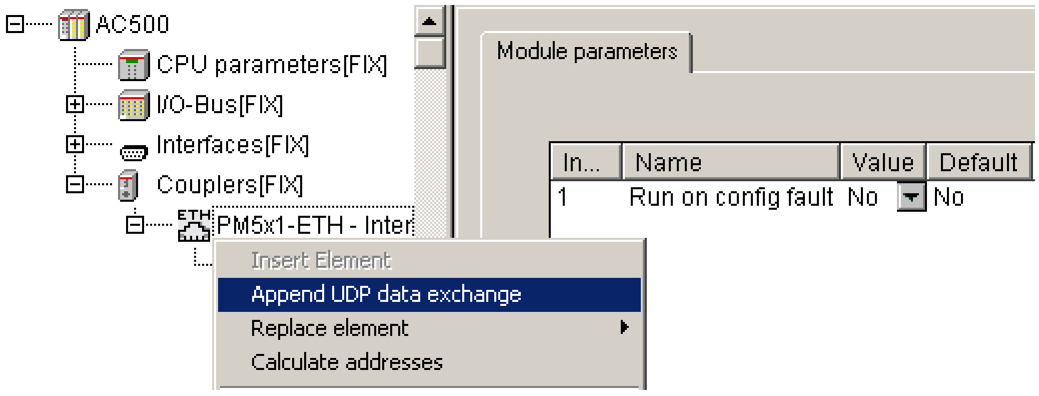


I...	Name	Value	Def...	Min.	Max.
3	Disable write to %MB0.x from	0	0	0	65535
4	Disable write to %MB0.x to	0	0	0	65535
5	Disable read to %MB0.x from	0	0	0	65535
6	Disable read to %MB0.x to	0	0	0	65535
7	Disable write to %MB1.x from	0	0	0	65535
8	Disable write to %MB1.x to	0	0	0	65535
9	Disable read to %MB1.x from	0	0	0	65535
10	Disable read to %MB1.x to	0	0	0	65535

Like for Modbus RTU on the serial interfaces COMx, it is also possible to disable read and/or write access to individual segments for slave operation. Reading/writing is disabled beginning at the set address and is valid up to the set end address (inclusive).

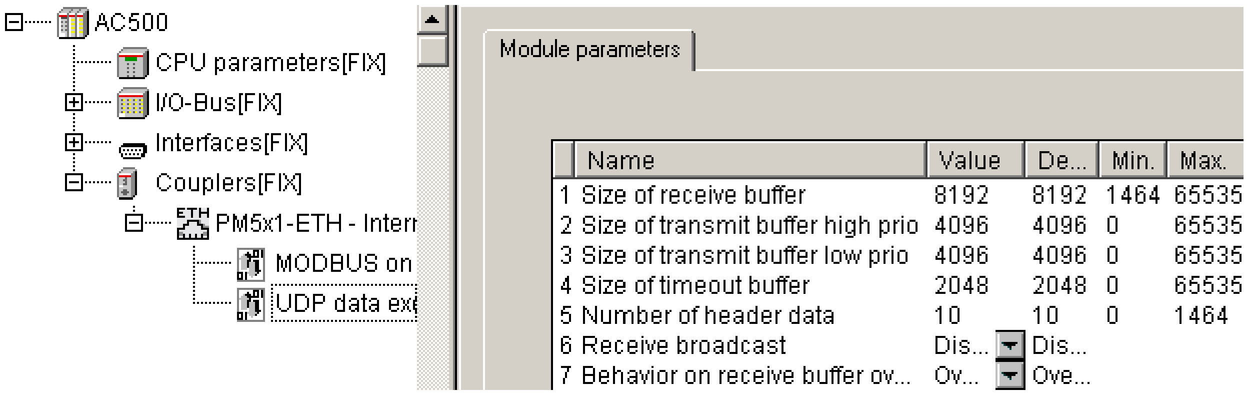
Parameter	Default value	Value	Meaning
Disable write to %MB0.x from	0	0...65535	Disable write access for segment 0 starting at %MB0.x
Disable write to %MB0.x to	0	0...65535	Disable write access for segment 0 up to %MB0.x
Disable read to %MB0.x from	0	0...65535	Disable read access for segment 0 starting at %MB0.x
Disable read to %MB0.x to	0	0...65535	Disable read access for segment 0 up to %MB0.x
Disable write to %MB1.x from	0	0...65535	Disable write access for segment 1 starting at %MB1.x
Disable write to %MB1.x to	0	0...65535	Disable write access for segment 1 up to %MB1.x
Disable read to %MB1.x from	0	0...65535	Disable read access for segment 1 starting at %MB1.x
Disable read to %MB1.x to	0	0...65535	Disable read access for segment 1 up to %MB1.x

The protocol "UDP data exchange" can be appended to the coupler by right-clicking the internal Ethernet coupler "PM5x1-ETH - Internal Ethernet" in the configuration tree and selecting the context menu item "Append UDP data exchange".



In...	Name	Value	Default
1	Run on config fault	No	No

The following parameters can be set for the "UDP data exchange" protocol:



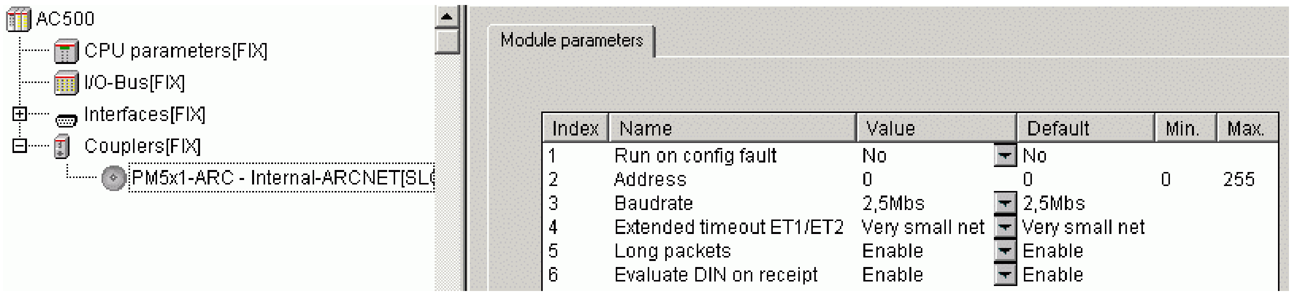
	Name	Value	De...	Min.	Max.
1	Size of receive buffer	8192	8192	1464	65535
2	Size of transmit buffer high prio	4096	4096	0	65535
3	Size of transmit buffer low prio	4096	4096	0	65535
4	Size of timeout buffer	2048	2048	0	65535
5	Number of header data	10	10	0	1464
6	Receive broadcast	Dis...	Dis...		
7	Behavior on receive buffer ov...	Ov...	Ove...		

Parameter	Default value	Value	Meaning
Size of receive buffer	8192	1464..65535	Size of receive buffer in bytes The minimum size is equal to the maximum size of an UDP telegram
Size of transmit buffer high prio	4096	0..65535	Size of transmit buffer (in bytes) for telegrams with high priority
Size of transmit buffer low prio	4096	0..65535	Size of transmit buffer (in bytes) for telegrams with low priority
Size of timeout buffer	2048	0..65535	Size of buffer (in bytes) for timeout data packages
Number of header data	10	0..1464	Number of header data to be copied to the timeout buffer for timeout packages (in bytes)
Receive broadcast	Disable	Disable	Reception of broadcast telegrams disabled (data packages to all stations)
		Enable	Reception of broadcast telegrams enabled (data packages to all stations)
Behavior on receive buffer overflow	Overwrite	Overwrite	Behavior on overflow of the receive buffer: The oldest data packages stored in the receive buffer are overwritten with the new incoming data packages.
		Reject	Behavior on overflow of the receive buffer: New incoming data are dismissed.

3.6.1.2 The internal ARCNET coupler PM5x1-ARCNET

Note: The controller with internal ARCNET coupler will be available as of version V1.2.

In the PLC configuration, the following parameters can be set for the internal ARCNET coupler "PM5x1-ARCNET - Internal ARCNET":



Index	Name	Value	Default	Min.	Max.
1	Run on config fault	No	No		
2	Address	0	0	0	255
3	Baudrate	2,5Mbs	2,5Mbs		
4	Extended timeout ET1/ET2	Very small net	Very small net		
5	Long packets	Enable	Enable		
6	Evaluate DIN on receipt	Enable	Enable		

Parameter	**Default value**	**Value**	**Meaning**
Run config fault	No	No	In case of a configuration error, the user program is not started.
		Yes	The user program is started independent of a faulty configuration of the internal ARCNET coupler.
Address	0	0...255	Address (node ID) of the ARCNET coupler
Baudrate see remark 1	2.5 MB/s	2.5 MB/s	Baudrate set for the ARCNET coupler
		1.25 MB/s	
		625 kB/s	
		312.5 kB/s	

Extended timeout ET1/ET2	Very small net	Very small net (=0)	ARCNET timeout setting. The following applies:
			Bit 0 configures ET2 of the coupler
			Bit 1 configures ET1 of the coupler
			
			`Value ET1 ET2  Meaning`
			`-----------------------`
			` 0     1   1    Max. network expansion 2 km`
			` 1     1   0`
			` 2     0   1`
			` 3     1   1    for large networks`

		Small net (=1)	
		Big net (=2)	
		Very big net (=3)	
Long packets	Enable	Enable	Enable long data packages (512 bytes)
		Disable	Incoming long data packages are received and dismissed. The SEND block indicates an error in case of long data packages.
Evaluate DIN on receipt see remark 2	Enable	Enable	Enable check of DIN identifier on receipt
		Disable	Disable check of DIN identifier on receipt

Remark 1: Baudrate of the ARCNET coupler

Note: If the baudrate set for the ARCNET coupler differs from the default value (2.5 MB/s), programming via ARCNET using the SoHard-ARCNET PC boards is no longer possible. The same baudrate has to be set for all subscribers of the ARCNET network. The ARCNET PC boards are firmly set to 2.5 MB/s.

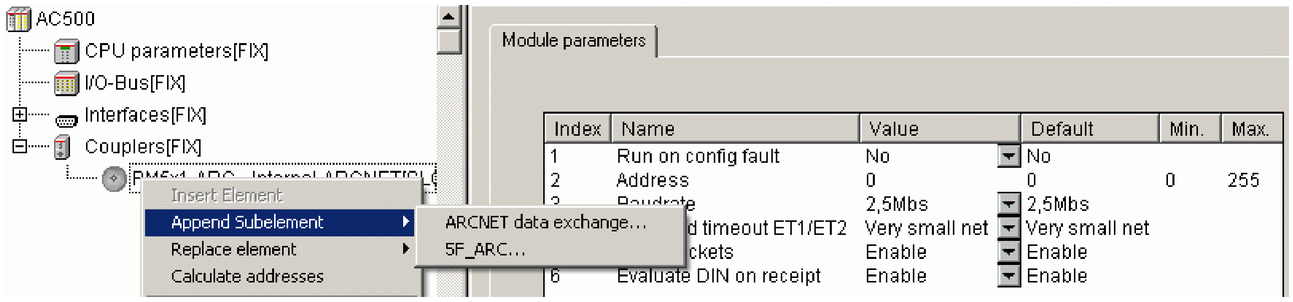
Remark 2: Check of DIN identifier on receipt

If the parameter "Evaluate DIN on receipt" is enabled (default setting), the following DIN identifiers are reserved:

DIN identifier		**Protocol**
Hex	**Dec**	
4F	79	"Online access" - Programming/OPC with CoDeSys / AC1131
5F	95	5F_ARCNET (MODBUS functions for ARCNET)
6F	111	PC331 programming (not used for AC1131/CoDeSys)
7F	127	Default DIN identifier for data exchange with function blocks: ARC_REC, ARC_SEND, ARC_STO, ARC_INFO All DIN identifiers except the reserved identifiers can be used for data exchange.

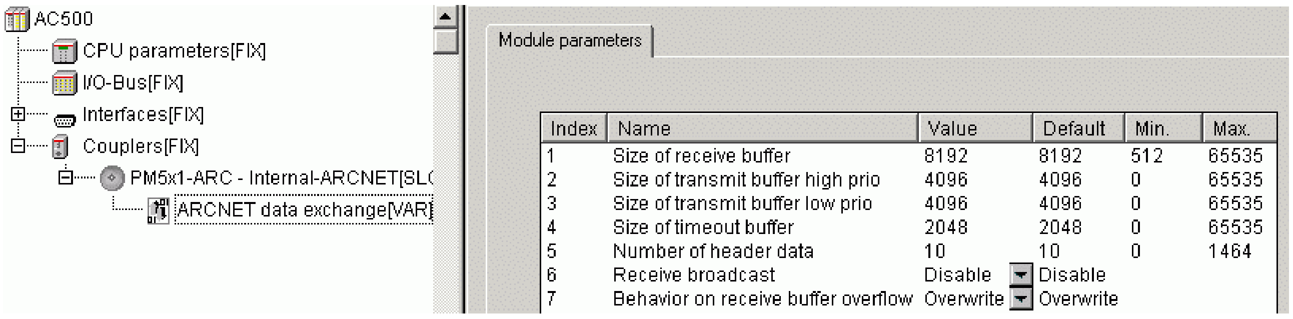
Caution: If the parameter "Evaluate DIN on receipt" is disabled, programming and/or OPC via ARCNET is not possible!

The protocols "ARCNET data exchange" and/or "5F_ARC" can be appended to the coupler by right-clicking the internal ARCNET coupler "PM5x1-ARC - Internal ARCNET" in the configuration tree and selecting the corresponding context menu item.



Index	Name	Value	Default	Min.	Max.
1	Run on config fault	No	No		
2	Address	0	0	0	255
3	Baudrate	2,5Mbs	2,5Mbs		
	d timeout ET1/ET2	Very small net	Very small net		
	ckets	Enable	Enable		
6	Evaluate DIN on receipt	Enable	Enable		

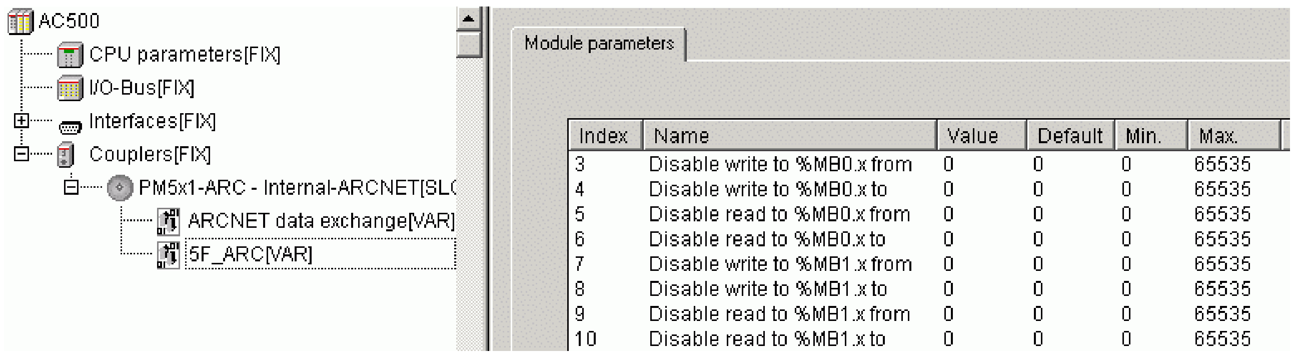
The following parameters can be set for the "ARCNET data exchange" protocol:



Index	Name	Value	Default	Min.	Max.
1	Size of receive buffer	8192	8192	512	65535
2	Size of transmit buffer high prio	4096	4096	0	65535
3	Size of transmit buffer low prio	4096	4096	0	65535
4	Size of timeout buffer	2048	2048	0	65535
5	Number of header data	10	10	0	1464
6	Receive broadcast	Disable	Disable		
7	Behavior on receive buffer overflow	Overwrite	Overwrite		

Parameter	**Default value**	**Value**	**Meaning**
Size of receive buffer	8192	512...65535	Receive buffer size in bytes. The minimum size is equal to the maximum size of an UDP telegram.
Size of transmit buffer high prio	4096	0...65535	Size of transmit buffer (in bytes) for telegrams with high priority.
Size of transmit buffer low prio	4096	0...65535	Size of transmit buffer (in bytes) for telegrams with low priority.
Size of timeout buffer	2048	0...65535	Size of buffer (in bytes) for timeout data packages.
Number of header data	10	0...1464	Number of header data to be copied to the timeout buffer for timeout packages (in bytes).
Receive broadcast	Disable	Disable	Reception of broadcast telegrams disabled (data packages to all stations).
		Enable	Reception of broadcast telegrams enabled (data packages to all stations).
Behavior on receive buffer overflow	Overwrite	Overwrite	Behavior on overflow of the receive buffer. The oldest data packages stored in the receive buffer are overwritten with the new incoming data packages.
		Reject	Behavior on overflow of the receive buffer. New incoming data are dismissed.

The following parameters can be set for the "5F_ARC" protocol:



Index	Name	Value	Default	Min.	Max.
3	Disable write to %MB0.x from	0	0	0	65535
4	Disable write to %MB0.x to	0	0	0	65535
5	Disable read to %MB0.x from	0	0	0	65535
6	Disable read to %MB0.x to	0	0	0	65535
7	Disable write to %MB1.x from	0	0	0	65535
8	Disable write to %MB1.x to	0	0	0	65535
9	Disable read to %MB1.x from	0	0	0	65535
10	Disable read to %MB1.x to	0	0	0	65535

Parameter	**Default value**	**Value**	**Meaning**
Disable write to %MB0.x from	0	0...65535	Disable write access for segment 0 starting at %MB0.x
Disable write to %MB0.x to	0	0...65535	Disable write access for segment 0 up to %MB0.x
Disable read %MB0.x from	0	0...65535	Disable read access for segment 0 starting at %MB0.x
Disable read %MB0.x to	0	0...65535	Disable read access for segment 0 up to %MB0.x
Disable write to %MB1.x from	0	0...65535	Disable write access for segment 1 starting at %MB1.x
Disable write to %MB1.x to	0	0...65535	Disable write access for segment 1 up to %MB1.x
Disable read %MB1.x from	0	0...65535	Disable read access for segment 1 starting at %MB1.x
Disable read %MB1.x to	0	0...65535	Disable read access for segment 1 up to %MB1.x

Note: For the AC500 CPU PM571, 4kB = %MB0.0 .. %MB0.4095 (i.e., not a complete segment) are available for the addressable flag area. Thus, not all Modbus addresses can be accessed.

3.6.2 Configuring the external couplers

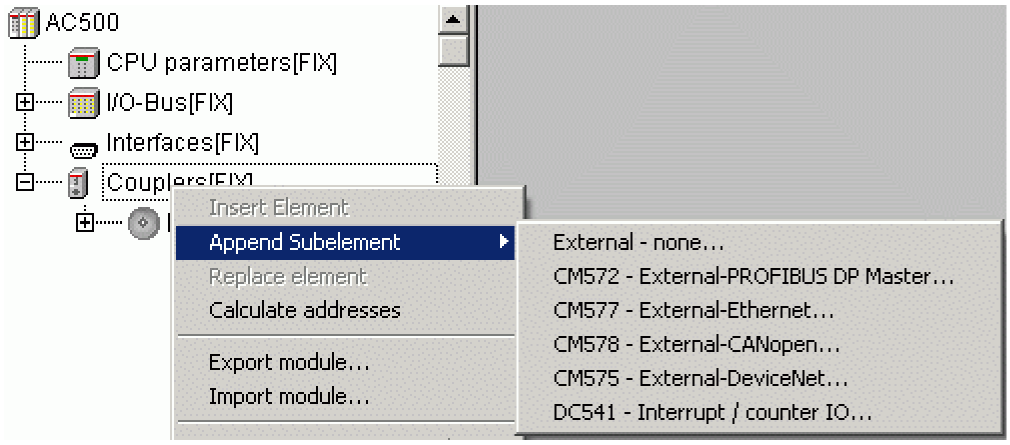
For an AC500 controller with external couplers, the couplers must be specified in the PLC configuration in the same order as they are installed in the hardware.

The following assignment between coupler line and slot applies to the couplers:

- Line 0 corresponds to the internal coupler (installed in the CPU's housing)
- Line 1 is the coupler installed in the first slot on the left of the CPU
- Line 2 is the coupler installed in the second slot on the left of the CPU
- Line 3 and 4 are installed in the third and fourth slot on the left of the CPU

The allocation of inputs/outputs for the couplers is done slot-oriented and independent of the coupler type.

To append the external coupler, right-click the element "Couplers" in the configuration tree and select the sub menu "Append Subelement". All available external couplers are shown:



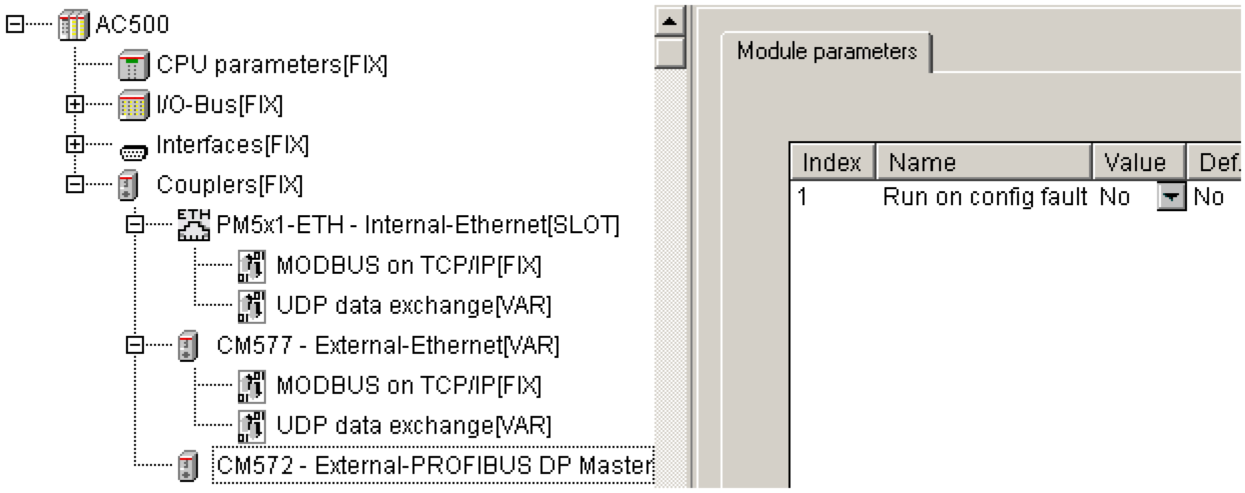
Note: The external couplers "CM578 - External CANopen", "CM575 - External DeviceNet" and the module "DC541 - Interrupt / counter IO" are available as of version V1.1.

For external couplers, the same applies as for internal couplers: only the behavior of the CPU in case of a coupler configuration error and the required protocols are set in the PLC configuration. The actual configuration of the external couplers is done with the integrated fieldbus configurator SYCON.net. For a detailed description of this procedure refer to the chapter "System Technology of the Couplers" (see also System Technology of the Couplers).

The following settings are possible:

Parameter	Default value	Value	Meaning
Run config fault	No	No	In case of a configuration error, the user program is not started.
		Yes	The user program is started independent of a faulty configuration of the particular external coupler.

The following figure shows an example of a PLC configuration consisting of an internal Ethernet coupler "PM5x1-ETH - Internal-Ethernet", an external Ethernet coupler "CM577 - External-Ethernet" and a PROFIBUS DP master coupler "CM572 - External-PROFIBUS DP Master":



Index	Name	Value	Def.
1	Run on config fault	No	No

PLCconf_Coupler6_E.gif

For the external Ethernet coupler "CM577 - External-Ethernet", the same parameters and protocol settings are possible as for the internal Ethernet coupler "PM5x1 - Internal-Ethernet" (see also Internal Ethernet coupler).

4 System start-up / program processing

4.1 Terms

Cold start:

- A cold start is performed by switching power OFF/ON if no battery is connected.
- All RAM memory modules are checked and erased.
- If no user program is stored in the Flash EPROM, the default values (as set on delivery) are applied to the interfaces.
- If there is a user program stored in the Flash EPROM, it is loaded into RAM.
- The default operating modes set by the PLC configuration are applied.

Warm start:

- A warm start is performed by switching power OFF/ON with a battery connected.
- All RAM memory modules are checked and erased except of the buffered operand areas and the RETAIN variables.
- If there is a user program stored in the Flash EPROM, it is loaded into RAM.
- The default operating modes set by the PLC configuration are applied.

RUN -> STOP:

- RUN -> STOP means pressing the RUN key on the PLC while the PLC is in RUN mode (PLC display "run").
- If a user program is loaded into RAM, execution is stopped.
- All outputs are set to FALSE or 0.
- Variables keep their current values, i.e., they are not initialized.
- The PLC display changes from "run" to "StoP".

START -> STOP:

- START -> STOP means stopping the execution of the user program in the PLC's RAM using the menu item "Online/Stop" in the programming system.
- All outputs are set to FALSE or 0.
- Variables keep their current values, i.e., they are not initialized.
- The PLC display changes from "run" to "StoP".

Reset:

- Performs a START -> STOP process.
- Preparation for program restart, i.e., the variables (VAR) are set to their initialization values.
- Reset is performed using the menu item "Online/Reset" in the programming system.

Reset (cold):

- Performs a START -> STOP process.
- Preparation for program restart, i.e., the variables (VAR) are set to their initialization values.
- Reset (cold) is performed using the menu item "Online/Reset (cold)" in the programming system.

Reset (original):

- Resets the controller to its original state (deletion of Flash, SRAM (%M, RETAIN), coupler configurations and user program!).
- Reset (original) is performed using the menu item "Online/Reset (original)" in the programming system.

STOP -> RUN:

- STOP -> RUN means pressing the RUN key on the PLC while the PLC is in STOP mode (PLC display "StoP").

- If a user program is loaded into RAM, execution is continued, i.e., variables will not be set to their initialization values.
- The PLC display changes from "StoP" to "run".

STOP -> START:

- STOP -> START means continuing the execution of the user program in the PLC's RAM using the menu item "Online/Start" in the programming system.
- If a user program is loaded into RAM, execution is continued, i.e., variables will not be set to their initialization values.
- The PLC display changes from "StoP" to "run".

Download:

- Download means loading the complete user program into the PLC's RAM. This process is started by selecting the menu item "Online/Download" in the programming system or after confirming a corresponding system message when switching to online mode (menu item "Online/Login").
- Execution of the user program is stopped.
- In order to store the user program to the Flash memory, the menu item "Online/Create boot project" must be called after downloading the program.
- Variables are set to their initialization values according to the initialization table.
- RETAIN variables can have wrong values as they can be allocated to other memory addresses in the new project!
- A download is forced by the following:

  - changed PLC configuration
  - changed task configuration
  - changed library management
  - changed compile-specific settings (segment sizes)
  - execution of the commands "Project/Clean all" and "Project/Rebuild All".

Online Change:

- After a project has changed, only these changes are compiled when pressing the key <F11> or calling the menu item "Project/Build". The changed program parts are marked with a blue arrow in the block list.
- The term Online Change means loading the changes made in the user program into the PLC's RAM using the programming system (after confirming a corresponding system message when switching to online mode, menu item "Online/Login").
- Execution of the user program is not stopped. After downloading the program changes, the program is re-organized. During re-organization, no further online change command is allowed. The storage of the user program to the Flash memory using the command "Online/Create boot project" cannot be initiated until re-organization is completed.
- Online Change is not possible after:

  - changes in the PLC configuration
  - changes in the task configuration
  - changes in the library management
  - changed compile-specific settings (segment sizes)
  - performing the commands "Project/Clean all" and "Project/Rebuild All".

Data buffering:

- Data buffering, i.e., maintaining data after power ON/OFF, is only possible, if a battery is connected. The following data can be buffered completely or in parts:
  - Data in the addressable flag area (%M area)
  - RETAIN variable
  - PERSISTENT variable (number is limited, no structured variables)
  - PERSISTENT area (%R area, as of V1.2)
- In order to buffer particular data, the data must be excluded from the initialization process.

4.2 Start of the user program

The user program (UP) is started according to the following table. Here it is assumed that a valid user program is stored to the Flash memory.

Action	**No SD card** with UP 1) installed, Auto run = **ON**	**No SD card** with UP 1) installed, Auto run = **OFF**	**SD card** with UP 1) installed, Auto run = **ON**	**SD card** with UP 1) installed, Auto run = **OFF**
Voltage ON or **Warm start** or **Cold start**	UP is loaded from Flash into RAM and started from Flash.	No UP is loaded from Flash. When logging in, the message "No program available in the controller ..." is displayed.	UP is loaded from the SD card into Flash memory and RAM and then started from RAM.	UP is loaded from the SD card to the Flash memory. RAM remains empty. When logging in, the message "No program available in the controller ..." is displayed.
STOP -> RUN	UP in RAM is started.	UP in RAM is started.	UP in RAM is started.	UP in RAM is started.
STOP -> START	UP in RAM is started.	UP in RAM is started.	UP in RAM is started.	UP in RAM is started.
Download 2)	The UP currently stored in the CPU's RAM is stopped. The built UP is loaded from the PC into the PLC's RAM.	The built UP is loaded from the PC into the PLC's RAM.	The UP currently stored in the CPU's RAM is stopped. The built UP is loaded from the PC into the PLC's RAM.	The built UP is loaded from the PC into the PLC's RAM.
Online Change 3)	Processing of the UP currently stored in the CPU's RAM is continued. The changes made to the UP are loaded from the PC into the PLC's RAM. The UP is re-organized and processed.	The changes made to the UP are loaded from the PC into the PLC's RAM. The UP is re-organized.	Processing of the UP currently stored in the CPU's RAM is continued. The changes made to the UP are loaded from the PC into the PLC's RAM. The UP is re-organized and processed.	The changes made to the UP are loaded from the PC into the PLC's RAM. The UP is re-organized.

Remarks:

1) The version of the PLC operating system used to generate the SD card and the version of the PLC operating system to which the UP is to be loaded from the SD card must be the same. If the versions differ, the SD card is not loaded.

2) After the download is completed, the program is not automatically stored to the Flash memory. To perform this, select the menu item "Online/Create boot project". If the UP is not stored to the Flash memory, the UP is reloaded from the Flash memory after voltage OFF/ON.

   The program is started either by pressing the RUN/STOP key or using the menu item "Online/Start" in the programming system.

3) After the Online Change process is completed, the program is not automatically stored to the Flash memory. To perform this, select the menu item "Online/Create boot project" after re-organization is completed. During re-organization and flashing, no further online change command is allowed.

   If the UP is not stored to the Flash memory, the UP is reloaded from the Flash memory after voltage OFF/ON.

4.3 Data backup and initialization

4.3.1 Initialization of variables, overview

The initialization of variables to 0 or to the initialization value is performed by switching voltage ON, by a reset or after downloading the user program.

If internal variables shall be buffered, these variables have to be marked as "VAR_RETAIN" or "VAR_RETAIN PERSISTENT". This applies to both the internal variables and the variables of the addressable flag area (%M area).

Note: The order of the internal RETAIN variables is only kept when using the online change command. If the program is re-built, the order can change and, due to this, the buffered variables do no match.
See also CoDeSys / Retentive variables, chapter "Behavior of RETAIN variables on download".

The following table shows an overview of the initialization values of the individual variableles:

Action → / Variable ↓	without battery unch. = unchanged										with battery unch. = unchanged									
	Voltage ON	STOP –> RUN (pushbutton)	STOP –> START (PC)	Download	Online Change	Reset	Reset (cold)	Voltage OFF/ON after Reset (cold)	Reset (origin), factory setting	Download after Reset (origin)	Voltage ON	STOP –> RUN (pushbutton)	STOP –> START (PC)	Download	Online Change	Reset	Reset (cold)	Voltage OFF/ON after Reset (cold)	Reset (origin), factory setting	Download after Reset (origin)
VAR	0	unch.	unch.	0	unch.	0	0	0	???	0	0	unch.	unch.	0	unch.	0	0	0	???	0
VAR := Value	value	unch.	unch.	value	unch.	value	value	value	???	value	value	unch.	unch.	value	unch.	value	value	value	???	value
VAR %MDx.x	0	unch.	unch.	0	unch.	0	0	0	???	0	0	unch.	unch.	0	unch.	0	0	0	???	0
VAR %MDx.x := Value	value	unch.	unch.	value	unch.	value	value	value	???	value	value	unch.	unch.	value	unch.	value	value	value	???	value
VAR_RETAIN	0	unch.	unch.	0	unch.	unch.	0	0	???	0	unch.	unch.	unch.	0	unch.	unch.	0	unch.	???	0
VAR_RETAIN := Value	0	unch.	unch.	value	unch.	unch.	value	0	???	value	unch.	unch.	unch.	value	unch.	unch.	value	unch.	???	value
VAR_RETAIN %MDx.x	0	unch.	unch.	0	unch.	unch.	0	0	???	0	unch.	unch.	unch.	0	unch.	unch.	0	unch.	???	0
VAR_RETAIN %MDx.x := Value	0	unch.	unch.	value	unch.	unch.	value	0	???	value	unch.	unch.	unch.	value	unch.	unch.	value	unch.	???	value
VAR_PERSISTENT	0	unch.	unch.	unch.	unch.	0	0	0	???	0	0	unch.	unch.	unch.	unch.	0	0	0	???	0
VAR_PERSISTENT := Value	value	unch.	unch.	unch.	unch.	value	value	value	???	value	value	unch.	unch.	unch.	unch.	value	value	value	???	value
VAR_PERSISTENT %MDx.x	0	unch.	unch.	unch.	unch.	0	0	0	???	0	0	unch.	unch.	unch.	unch.	0	0	0	???	0
VAR_PERSISTENT %MDx.x := Value	value	unch.	unch.	unch.	unch.	value	value	value	???	value	value	unch.	unch.	unch.	unch.	value	value	value	???	value
VAR_RETAIN PERSISTENT	0	unch.	unch.	unch.	unch.	unch.	0	0	???	0	unch.	unch.	unch.	unch.	unch.	unch.	0	unch.	???	0
VAR_RETAIN PERSISTENT := Value	0	unch.	unch.	unch.	unch.	unch.	value	0	???	value	unch.	unch.	unch.	unch.	unch.	unch.	value	unch.	???	value
VAR_RETAIN PERSISTENT %MDx.x	0	unch.	unch.	unch.	unch.	unch.	0	0	???	0	unch.	unch.	unch.	unch.	unch.	unch.	0	unch.	???	0
VAR_RETAIN PERSISTENT %MDx.x := Value	0	unch.	unch.	unch.	unch.	unch.	value	0	???	value	unch.	unch.	unch.	unch.	unch.	unch.	value	unch.	???	value

4.3.2 Notes regarding the declaration of retentive variables and constants

To guarantee the correct initialization or backing up of variables according to the table shown above, the following rules have to be observed when declaring variables.

Declaration of retentive internal variables:

The variables have to be declared as **VAR_RETAIN** or **VAR_GLOBAL RETAIN**.

Example:

```
(* Declaration in the global variable lists *)

VAR_GLOBAL RETAIN
            byVar : BYTE;
            wVar : WORD;
            rVar : REAL;
END_VAR

(* Declaration in the program *)

VAR RETAIN
            byVar1 : BYTE;
END_VAR
```

Declaration of retentive variables in %M area:

The variables have to be declared as **VAR_RETAIN** or **VAR_GLOBAL RETAIN**.

Note: As of Control Builder version V1.2 and firmware version V1.2.0, a new persistent area is available which can be buffered by a specific setting in the PLC configuration
(see also chapter "The addressable PERSISTENT area").

Declaration of constants:

Constants are declared as **VAR_GLOBAL CONSTANT** or **VAR_CONSTANT**.

Example:

```
(* Declaration as global constants *)

VAR_GLOBAL CONSTANT
            byConst_1 : BYTE := 1;
END_VAR

(* Declaration in the program *)

VAR CONSTANT
            byConst_2 : BYTE := 2;
END_VAR
```

Note: Using the option "Replace constants" available under "Project" => "Options" => "Build", it is possible to specify whether constants are treated as variables (i.e., writing the variable is possible) or the value is directly entered into the code when building the project.

4.4 Processing times

4.4.1 Terms

The most important times for the use of the AC500 basic unit with or without connected remote modules are:

- The **reaction time** is the time between a signal transition at the input terminal and the signal response at the output terminal.
  For binary signals, the reaction time is composed of the input delay, the cycle time of the program execution and the bus transmission time if the system is expanded by remote modules.

- The **cycle time** determines the time intervals after which the processor restarts the execution of the user program.
  The cycle time has to be specified by the user. It should be greater than the program processing time of the user program plus the data transfer times and the related waiting times.
  The cycle time is also the time base for some time-controlled functions, such as for the INTK.

- The **program processing time** is the net time for processing the user program.

4.4.2 Program processing time

Statements	PM57x	PM58x	PM59x
- Binary statements of the type:			
!M /M &M =M !NM /NM &NM =NM !M /M &M =SM !NM /NM &NM =RM Processing time for 1000 statements:	0.3 ms	0.15 ms	0.02 ms
- Word statements of the type:			
!MW +MW -MW =MW !-MW -MW +MW =-MW !MW *MW :MW =MW !-MW *-MW :-MW =-MW Processing time for 1000 statements:	0.3 ms	0.15 ms	0.01 ms
- Floating point:			
Processing time for 1000 statements:	6 ms	3 ms	0.02 ms

4.4.3 Set cycle time

It is assumed that the processor always gets access in a moment with a worst-case condition.

The cycle time is stored in the task configuration and can be selected in steps of 1 ms. If the selected cycle time is too short, the processor will not be able to completely process the tasks assigned to it every cycle. This will result in a time delay.

If this lack of time becomes too large over several cycles, the processor aborts the program execution and outputs an error (E2).

For some function blocks, such as the PID controller, the error-free execution depends on an exact timing sequence. Make sure that there is a larger time reserve.

To check the correct cycle time, perform the following steps:

- Load the user program into the basic unit.

- Check the capacity utilization with "Online/PLC Browser/cpuload".

- Change the cycle time until the capacity utilization is below 80 %.

When setting the cycle time, consider the following values:

- Time for reading and copying the input signals from the I/O driver to the I/O image.
- Time for copying the input signals of the user task from the I/O image to the image memory.
- Program processing time
- Time for copying the output signals of the user task from the image memory to the I/O image.
- Time for copying the output signals from the I/O image to the I/O driver and applying the I/Os to the I/O module.
- Receiving/sending interrupts from coupler telegrams within the cycle time.
- Receiving/sending interrupts from the serial interface within the cycle time.
- Task changes.
- Runtime of the watchdog task.

4.5 Task configuration for the AC500 CPU

How to use the task configuration for the Control Builder is described in detail in the chapter "Resources / Task configuration" (see also 3S: CoDeSys Programming System / Resources / Task configuration).

This section describes the specialities for the task configuration for the AC500.

The possible number of tasks depends on the CPU type. For PM571 and PM581, a maximum of 3 user tasks can be created, for PM591 a maximum of 16 user tasks is allowed.

If **no task configuration** is specified in the project, a task with the following properties is created automatically.

Type = cyclic
Priority = 10
Cycle time = t#10ms
Program call= PLC_PRG.

In version 1.0, the following task types are possible for the AC500 CPU: **"cyclic"** and **"free running"**. The types "event triggered" and "external event triggered" are not possible.

As of Control Builder version V1.1 and firmware version V1.1.7, also the task type **"external event triggered"** can be selected for the interrupt and counting device DC541 (see also System Technology DC541 - Interrupt-Mode).

All 32 priorities can be selected for the user tasks, where 0 is the highest priority and 31 the lowest. The default priority is 10.

Priorities **lower than 10** are reserved for high-priority processes with a **very short program execution time**. The priorities 10 to 31 are intended for "normal" user tasks or tasks with a long program execution time.

⚠ **Caution:** Using, for example, a priority lower than 10 for a task with a long program execution time can cause, for example, the CS31 bus and/or the FBP interface to fail.

5 The diagnosis system in the AC500

5.1 Summary of diagnosis possibilities

5.1.1 Structure of the diagnosis system

The AC500 contains a diagnosis system that allows to manage up to 100 error messages in a circular buffer. For each of these events, a time stamp with date and time based on the controller's real-time clock (RTC) is generated in the runtime system. The time stamp consists of three entries:

- Error occurrence (come)
- Error disappearance (gone)
- Error acknowledgement

If no battery is insterted in the PLC, the PLC clock is set to the following value when switching on the control voltage:

01. January 1970, 00:00

Each error message has a unique error number. This number provides the following information:

- State (come, gone, acknowledged)
- Error class
- Faulty component
- Faulty device
- Faulty module
- Faulty channel
- Error identifier

For a description of the error numbers, please refer to section Organization and structure of error numbers later in this chapter.

The error numbers are divided into the following error classes:

Class	Type	Description	Example
E1	Fatal error	Safe operation of the operating system is no longer ensured.	Checksum error in system Flash, RAM error
E2	Serious error	The operating system works correctly, but the error-free execution of the user program is not ensured.	Checksum error in user Flash, task cycle times exceeded
E3	Light error	It depends on the application whether the user program has to be stopped by the operating system or not. The user decides which reaction is to be done.	Flash memory cannot be programmed, I/O module failed
E4	Warning	Errors that occur on peripheral devices or that will have an effect only in the future. The user decides which reactions are to be done.	Short circuit in an I/O module, battery empty/not installed

There are different possibilities to access the error messages:

- Diagnosis directly at the PLC by means of "ERR" LED, keypad and display
- Plain-text display of the error messages in the status line of the Control Builder in online mode
- Diagnosis with the PLC browser commands of the Control Builder
- Diagnosis with help of the user program using the diagnosis blocks of the library SysInt_AC500_Vxx.LIB

5.1.2 Diagnosis directly at the PLC by means of "ERR" LED, keypad and display

If the PLC contains a non-acknowledged error, the red error LED "ERR" lights up.

Note: The CPU parameter "Error LED" in the PLC configuration allows to set for which error class the LED indicates an error. The default setting is "On", i.e., errors of all error classes are indicated. If the parameter is set for example to "Off_by_E3", the LED "ERR" does not light up in case of errors of the classes E3 and E4. However, if an E2 error occurs, the LED always lights up.
See also chapter "Configuration of the CPU parameters".

If one or several non-acknowledged errors exist, the errors can be displayed and acknowledged according to their occurrence order using the <DIAG> key. Pressing the <DIAG> key the first time displays the error class and error identifier. After this, pressing the <DIAG> key several times browses through the detail information d1=component, d2=device, d3=module and d4=channel. If the "d4" information is displayed and the <DIAG> key is pressed once more, the error class/error identifier is re-displayed.

If you quit the diagnostic display by pressing <ESC>, the error is not acknowledged and displayed again when pressing the <DIAG> key.

If you quit the diagnostic display with the <QUIT> key, the error is acknowledged.

The LED "ERR" goes off when all errors are acknowledged.

Example:

The error "Battery empty or not installed" appears in the PLC display as follows:

Pushbutton	Display	Meaning
<DIAG>	E4 008	E4=Warning / Identifier = Empty/Not available
<DIAG>	d1 009	Detail information d1 = 009 -> Component=CPU
<DIAG>	d2 022	Detail information d2 = 022 -> Device=Battery
<DIAG>	d3 031	Detail information d3 = 031 -> Module=no specification
<DIAG>	d4 031	Detail information d4 = 031 -> Channel=no specification
<DIAG>	E4 008	E4=FK4 / Identifier = Empty/Not available
<ESC>	run/StoP	Diagnostic display is quit without error acknowledgement.
<DIAG>	E4 008	E4=FK4 / Identifier = Empty/Not available
<QUIT>	run/StoP	Diagnostic display is quit with error acknowledgement. If no further non-acknowledged errors exist, the LED "ERR" goes off.

For a description of the error numbers refer to the chapter Organization of the error numbers.

5.1.3 Plain-text display of error messages in the Control Builder status line during online mode

When the Control Builder is switched to online mode, incoming error messages or status changes of an error message (come, gone, acknowledged) are displayed as plain-text in the status line. For example, the acknowledgment of the error message "Battery empty or not installed" described in the previous section is displayed in online mode as follows:

```
#152502216: x 1970-01-01 06:33:53 E4 :' No battery or battery empty
```

With:

#152502216	Online error number
x	Error acknowledged (+ = Error come, - = Error gone)
1970-01-01 06:33:53	Time stamp (time of acknowledgement)
E4 :	Error class 4 = Warning
No battery or battery empty	Error text (according to language setting for the Control Builder)

Note: The error text is read from the file Errors.ini and displayed according to the online error number. The language of the error text depends on the language setting for the Control Builder. Errors which do not have an entry in the file Errors.ini are displayed without error text. The file Errors.ini is part of the Target Support Package (TSP) and located in the directory ..\Targets\ABB_AC500 or ..\Targets\ABB_AC500\AC500_V12.

5.1.4 Diagnosis using the PLC browser commands of the Control Builder

All errors or errors of a certain error class can be displayed and/or acknowledged using the PLC browser of the Control Builder. Also the complete diagnosis system can be deleted.

The PLC browser commands are described in detail in the chapter AC500-specific PLC browser commands.

5.1.5 Diagnosis with help of the user program

The entries in the diagnosis system can also be accessed from the user program with the help of specific diagnosis blocks. These blocks are described in the chapter The diagnosis blocks of the AC500.

5.2 Organization and structure of error numbers

For each error, an error number is stored in the firmware. This error number is coded as follows:

Status	Error class	Faulty component	Faulty device	Faulty module	Faulty channel	Error identifier
	28...29	**24...27**	**16...23**	**11...15**	**6...10**	**0...5**
4 bits	2 bits	4 bits	8 bits	5 bits	5 bits	6 bits
	0...3	0...15	0...255	0...31	0...31	0...63

Status value	Meaning
Bit 0	not used
Bit 1	Error occurrence (come)
Bit 2	Error removed (gone)
Bit 3	Error acknowledgement

In addition to the error information, the diagnosis message also contains status information (1 bit per status). Each status is set by a specific event:

- Error occurrence (come)
- Error acknowledgement
- Error removal (gone)

The diagnosis message is generated when an error occurs. In this case, the status bit 1 is set. If this error is acknowledged or removed afterwards, the corresponding status bits are set additionally.

In the Control Builder, the **online error number** is displayed. This error number is sent to the Control Builder when working in online mode and decoded **language-dependent** using the controller description file **Errors.ini**. The error number is coded as follows:

Faulty component	**Faulty device**	**Faulty module**	**Faulty channel**	**Error identifier**
24...27	16...23	11...15	6...10	0...5
4 bits	8 bits	5 bits	5 bits	6 bits
0...15	0...255	0...31	0...31	0...63

The error status (come, gone, acknowledged) and the error class are hidden in the online error number and displayed as plain-text.

5.2.1 Error classes

The error classes are coded as follows:

Value error class			**Meaning**
Bit 29	**Bit 28**	**Class**	
0	0	1	E1 fatal errors
0	1	2	E2 serious errors
1	0	3	E3 light errors
1	1	4	E4 warnings

5.2.2 Error identifiers

The following error identifiers are defined. The identifiers are kept generally in order to reach a maximum systematic. The exact meaning of each error depends on the further information provided by the error messages. For example, the error identifiers 'Highest level' and 'Lowest level' for analog channels correspond to the 'Out of Range' message.

Error Identifier		
Err	Meaning	Meaning
0	Fehler allgemein	General
1	Fehler falscher Wert	Wrong value
2	Fehler ungültiger Wert	Invalid value
3	Fehler Timeout	Timeout
4	Fehler oberster Grenzwert	Highest level
5	Fehler oberer Grenzwert	High level
6	Fehler unterer Grenzwert	Low level
7	Fehler unterster Grenzwert	Lowest level
8	Fehler leer/fehlt	Empty or missing
9	Fehler voll	Full
10	Fehler zu groß	Too big
11	Fehler zu klein	Too small
12	Fehler beim Lesen	Read
13	Fehler beim Schreiben	Write
14	Fehler beim Löschen	Delete
15	Fehler beim Reservieren von Speicher	Alloc memory
16	Fehler beim Freigeben von Speicher	Free memory
17	Fehler beim Zugriff	Access
18	Fehler beim Testen	Test
19	Fehler Checksumme	Checksum
20	Fehler Message	Message
21	Fehler bei PutMessage	Put message
22	Fehler bei GetMessage	Get message
23	Fehler Warten auf freie Message	Wait message
24	Fehler Message gelöscht	Message deleted
25	Fehler Warten auf Antwort	Wait answer
26	Fehler Konfiguration	Config data
27	Fehler keine Konfiguration	No config
28	Fehler Unterschied Soll-/Ist-Konfiguration	Different config
29	Fehler beim Schreiben der Konfiguration	Write config
30	Fehler beim Lesen der Konfiguration	Read config
31	Fehler anderer Typ / anderes Modell	Wrong type or model
32	Fehler unbekannter Typ / unbekanntes Modell	Unknown type or model
33	Fehler WaitReset	Wait reset
34	Fehler WaitReady	Wait ready
35	Fehler WaitRun	Wait run
36	Fehler WaitCom	Wait com
37	Fehler Zykluszeit	Cycle time
38	Fehler Exception	Exception
39	Fehler unbekannter Baustein	Unknown POU
40	Fehler Version	Version
41	Fehler Übertragung	Transmit
42	Fehler Empfang	Receive
43	Fehler intern	Internal
44	Fehler keine Abgleichwerte	No adjustment values
45	Drahtbruch	Cut wire
46	Überlast	Overload
47	Kurzschluss	Short circuit
48	Überlast / Drahtbruch	Overload / Cut wire
49	Kurzschluss / Drahtbruch	Short-circuit / Cut wire
50	Überlast / Kurzschluss	Overload / Short-circuit
51	Überlast / Kurzschluss / Drahtbruch	Overload / Short-circuit / Cut wire
63 (max.)	weitere	others

5.2.3 Possible error numbers

The following tables contain the possible combinations of error numbers.

Component		Device		Module or type		Channel		Remark
No.	Comp	No.	Dev	No.	Mod	No.	Ch	<- PLC browser
	d1		d2		d3		d4	<- Display
9	CPU	0	CPU	1	Operating system	1	Initialization error	
						2	Runtime error	
						3	Configuration error	
						31	Operating system	
				2	Runtime system	1	Initialization error	
						2	Runtime error	
						3	Configuration error	
						31	Operating system	
				4	IEC task online display %s	1	Initialization error	
						2	Runtime error	
		1	External coupler 1...6 or internal	1	Initialization	0	not used	
				2	Runtime error			
				3	Configuration			26,
				4	Protocol			
				31	Coupler			
		2	External coupler 2	1	Initialization	0	not used	
				2	Runtime error			
				3	Configuration			
				4	Protocol			
				31	Coupler			
		3	External coupler 3	1	Initialization	0	not used	
				2	Runtime error			
				3	Configuration			26,
				4	Protocol			
				31	Coupler			
		4	External coupler 4	1	Initialization	0	not used	
				2	Runtime error			
				3	Configuration			26,
				4	Protocol			
				31	Coupler			
		10	Internal coupler	1	Initialization	0	not used	
				2	Runtime error			
				3	Configuration			26,
				4	Protocol			
				31	Coupler			
		11	COM1	1	Initialization	0	not used	
				2	Runtime error			
				3	Configuration			26,
				4	Protocol			
				31	COM			
		12	COM2	1	Initialization	0	not used	
				2	Runtime error			
				3	Configuration			26,
				4	Protocol			
				31	COM			
		13	FBP	1	Initialization	0	not used	
				2	Runtime error			
				3	Configuration			26,
				4	Protocol			
				31	FBP			

		14	I/O bus	1	Initialization	0	not used	18, 15
				2	Runtime error			
				3	Configuration			
				4	Protocol			
				31	I/O bus			
		16	System EPROM	0...31	Sector, block no. or similar	0...31	Sector, block no. or similar (%s)	
		17	RAM	0...31		0...31		
		18	Flash EPROM	0...31		0...31		
		19	HW watchdog	31	Watchdog	31	Watchdog	
		20	SD Memory Card	1	Initialization	0...31	Sector, block no. or similar (%s)	
				2	Runtime error	0...31		
				3	Configuration	0...31		
				4	Protocol	0...31		
				31	SD card	0...31		
		21	Display	1	Initialization	0	not used	
				2	Runtime error			
				4	Protocol			
				31	Display			
		22	Battery	31	Battery	31	Battery	8,
		23	Clock	1	Initialization	0	not used	
				2	Runtime error			
				3	Configuration			
				4	Protocol			
				31	Clock			

Component		Device		Module or type		Channel		Remark
No.	Comp	No.	Dev	No.	Mod	No.	Ch	<- PLC browser
	d1		d2		d3		d4	<- Display
1..4 10	External coupler 1..4 or internal	0..254	Address/ Socket: Fieldbus: Slave ARCNET: ID partner - 1 Modbus: Comm partner	0..29	Module number	0..31	Channel number	
				30	Module number > 29	0..31	Channel number	
				31	Slave device	31	Slave device	
		255	Coupler	1	Initialization	0	not used	
				2	Runtime error			
				3	Configuration			
				4	Protocol			
				5	Operating system coupler	0...15	Bit 4...7 of the error number reported by the coupler see chapter "Coupler errors"	0...15 Bit 0...3 of the error number reported by the coupler see chapter "Coupler errors"
				6	Task 1 coupler			
				7	Task 2 coupler			
				8	Task 3 coupler			
				9	Task 4 coupler			
				10	Task 5 coupler			
				11	Task 5 coupler			
				12	Task 7 coupler			
				13	Watchdog coupler			
				31	Coupler			
11	COM1	0..254	Address: CS31: Slave Dec. expansion: Slave Modbus: Comm partner	0..29	Module number CS31: Module type: 00 - Digital input 01 - Analog input 02 - Digital output 03 - Analog output 04 - Digital in/output 05 - Analog in/output	0..31	Channel number	8, 48
				30	Module number > 29	0..31	Channel number	
				31	Slave device	31	Slave device	
		255	COM	1	Initialization	0	not used	
				2	Runtime error			
				3	Configuration			
				4	Protocol			
				31	COM			
12	COM2	0..254	Address: CS31: Slave Dec. expansion: Slave Modbus: Comm partner	0..29	Module number	0..31	Channel number	
				30	Module number > 29	0..31	Channel number	
				31	Slave device	31	Slave device	
		255	COM	1	Initialization	0	not used	
				2	Runtime error			
				3	Configuration			
				4	Protocol			
				31	COM			

Component		Device		Module or type		Channel		Remark
No.	Comp	No.	Dev	No.	Mod	No.	Ch	<- PLC browser
	d1		d2		d3		d4	<- Display
13	FBP	0... 254	Module number Parameter number	0..30	Slot number	0..31	Channel number	
		255	FBP	1	Initialization	0	not used	
				2	Runtime error			
				3	Configuration			
				4	Protocol			
				31	FBP			
14	I/O bus	0... 254	I/O bus module	0..6	Module type: 00 - Digital input 01 - Analog input 02 - Digital output 03 - Analog output 04 - Digital in/output 05 - Analog in/output 06 - others (e.g., fast counter)	0..31	Channel number	%s
				31	Module	1	Initialization error	
						2	Runtime error	
						3	Configuration	26,
						4	Protocol	
						31	Module	
		255	I/O bus	1	Initialization	0	not used	
				2	Runtime error			
				3	Configuration			
				4	Protocol			
				31	I/O bus			
15	User	0... 255	any	0..31	any	0..31	any, meaning is project-specific	

5.2.4 List of all errors

E1..E4	d1	d2	d3	d4	Identifier 000...063	AC500 display	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**	**<- displayed in 5)**
Byte 6 Bit 6..7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0..5**	**FBP-diagnosis block**	
Class	**Inter-face**	**De-vice**	**Mod-ule**	**Chan-nel**	**Error identifier**	**Error message**	**Remedy**
	1)	**2)**	**3)**	**4)**			
AC500 CPU errors							
Errors directly reported by the CPU							
Serious errors							
2	9	0	2	0	15	Not enough memory to generate the external reference list	
2	9	0	2	2	37	Cycle time is greater than the set watchdog time	Change task configuration
2	9	0	2	2	38	Access violation by an IEC task (for example zero pointer) (detail in call hierarchy as of V1.2)	Correct program
2	9	1..4/10	1	0	15	Watchdog task could not be installed	Check coupler
2	9	1..4/10	1	0	15	Installation of the coupler bus driver failed	Check coupler
2	9	1..4/10	1	0	15	Initialization error, not enough memory	- Check coupler - Check CPU FW version
2	9	1..4/10	1	0	17	Accessing test to the coupler failed	- Check coupler - Check CPU FW version
2	9	1..4/10	1	0	18	Watchdog test for the coupler failed	Check coupler
2	9	1..4/10	1	0	34	Timeout when setting the warm start parameters of the coupler	Check coupler
2	9	1..4/10	1	0	38	Installation of the coupler driver failed	Check coupler and FW version
2	9	1..4/10	2	0	15	Error occurred when creating the I/O description list of the coupler	Check coupler and FW version
2	9	1..4/10	4	0	15	Installation of a driver for the coupler failed	Check coupler and FW version
2	9	1..4/10	31	0..7	3	Watchdog error coupler/channel	Check coupler and FW version
2	9	11..13	1	0	15	Installation of a protocol driver for the serial interface failed, not enough memory	Check CPU FW version
2	9	11..13	1	0	17	Initialization of the protocol driver for the serial interface failed	Check CPU FW version
2	9	11..13	1	0	38	Installation of the hardware for the serial interface failed	Check CPU FW version
2	9	14	1	0	15	Not enough resources for the I/O-Bus	Check CPU FW version
2	9	14	1	0	17	Installation of the I/O-Bus driver failed	Check CPU FW version
2	9	14	1	0	43	Incorrect data format of the hardware driver of the I/O-Bus	Check CPU FW version
2	9	24	2	1	38	FPU division by zero	Clear up program
2	9	24	2	2	38	FPU overflow	Clear up program
2	9	24	2	3	38	FPU underflow	Clear up program
2	9	24	2	4	38	Forbidden FPU operation (e.g. 0/0)	Clear up program
2	9	24	2	6	38	FPU library function generated	Clear up program
Light errors							
3	9	0	2	1	17	Registering the handler for persistent data areas, areas do not work correctly	Check CPU FW version
3	9	0	2	2	37	User program contains an endless loop, a stop by hand is necessary	Correct user program
3	9	0	2	3	26	Configuration error	Adapt PLC configuration

3	9	0	4..31	3	26	Event-controlled task, unknown external event	Check task configuration
3	9	1..4/10	2	0	12	Error occurred when reading the I/O description	Check coupler configuration
3	9	1..4/10	2	0	26	Program was not started because of invalid configuration data of the coupler	Configure coupler
3	9	1..4/10	2	0	29	Error ucurred when deleting the configuration	Check coupler
3	9	1..4/10	3	0	26	In the PLC configuration, the coupler was not configured correctly or not at all	Adapt PLC configuration
3	9	11..13	3	0	26	Configuration error of the serial interface	Check configuration
3	9	14	3	0	26	Parameter "Error LED"=Failsafe is only allowed, if parameter "Behaviour of outputs in stop=actual state in hardware and online"	Change configuration
3	9	16	1	0	13	Deleting of the boot project failed (possibly no valid boot project available)	Reload project
3	9	21	31	0	17	Display could not be installed	Check FW
Warnings							
4	9	0	2	2	20	Program not started because of an existing error (see PLC configuration CPU parameters, stop on error class)	Eliminate error and acknowledge
4	9	0	2	2	37	Cycle time exceeded, but shorter than watchdog time	Adapt task configuration
4	9	0	2	2	43	Control system was restarted by FK2 or power dip according to PLC configuration	
4	9	0	3	1	40	Boot code versions V1.1.3. (or older versions) support smaller RAM disk	Update boot code to 1.2.0
4	9	1..4/10	1	0	32	Installation of the coupler driver failed, unknown coupler type	Check configuration
4	9	1..4/10	2	0	3	Within the specified time, connection could not be established to all of the slaves, I/O data may be (partly) invalid	Check slave for existence or set times according to the slave behaviour
4	9	1..4/10	2	0	4	No socket available	Check coupler settings with PLC browser
4	9	18	1	0	17	PLC Config file could not be read	Reload project
4	9	18	2	0	8	Error firmware update of SD card, file could not be opened	Check SD card, e.g. removed without "ejected"
4	9	20	1	2	2	Invalid value for FunctionOfCard in SDCARD.INI, value will be ignored	Check file SDCARD.INI on the SD card
4	9	20	1	10..14	8	Error Firmware update of the SD card, file could not be opened	Check SD card
4	9	20	1	10..14	12	Error Firmware update of the SD card, error while reading the file	Check SD card
4	9	20	1	10..14	13	Error Firmware update of the SD card, error while writing the file	Check CPU + coupler
4	9	20	1	10..14	17	Error Firmware update of the SD card, error while accessing the coupler	Check coupler
4	9	20	1	10..14	31	Error Firmware update of the SD card, file does not match the coupler type	Check SD card
4	9	20	3	20..24	8	Error while reading/writing the configuration data from/to the SD card, file could not be opened	Check SD card
4	9	20	3	20..24	12	Error while reading/writing the configuration data from/to the SD card, error while reading the file	Check SD card
4	9	20	3	20..24	13	Error while reading/writing the configuration data from/to the SD card, error while writing the file	Check SD card
4	9	20	3	20..24	17	Error while reading/writing the	Check SD card and

						configuration data from/to the SD card, error while accessing the coupler	hardware configuration
4	9	20	3	20..24	31	Error while reading/writing the configuration data from/to the SD card, file does not match the coupler type	Check SD card and hardware configuration
4	9	20	31	0..15	0..15	Coupler update failed, error message of the coupler: Channel = Bit 4..7 Error = Bit 0..3	see table "Coupler errors"
4	9	20	31	1	2	File does not exist	Check SD card
4	9	20	31	1	8	Error: Invalid file	Check SD card
4	9	20	31	1	40	Version is not supported, e.g. obsolete software	Check FW version, update to latest version
4	9	20	31	1	43	Other error, e.g. not enough memory or file system error	Update CPU FW
4	9	20	31	2	12	No SD card inserted or SDCARD.INI file not found	Check SD card
4	9	20	31	3	13	SD card errors: - SDCARD.INI on SD card is missing, default was generated - Copying the boot project from the SD card failed - Copying the boot project from the SD card failed (may be that there is no valid boot project) - Creating of the boot project failed (may be that there is no valid boot project)	Check SD card
4	9	20	31	5	13	Loading the source code failed	Check SD card, reload
4	9	20	31	31	8	Missing or exhausted battery	Insert battery or set parameter "Check Battery" to "Off"
Error messages of the I/O-Bus							
Serious errors							
2	14	1..10	31	1	34	Timeout while initializing an I/O module	Replace module
2	14	1..10	31	4	42	Failure of the module, more than the max. permissible communication errors have occurred in sequence	Check module, FW version (PLC-browser: IO-bus desc)
Light errors							
3	14	1..10	31	1	32	Master and module could not agree on any protocol variant, no variant found which is supported by both the master and the module	Check FW version CPU / I/O module
3	14	1..10	31	3	26	Configuration error PLC configuration master	Check configuration
3	14	255	2	0	3	Timeout while updating the I/O data at the program start	Check FW version CPU / I/O modules
3	14	255	2	0	26	Program was not started because of configuration error of the I/O-Bus	Check configuration
2	14	255	3	0	26	Configuration error PLC configuration master	Check configuration
Warnings							
4	14	1..10	31	1	34	Timeout during parameterization	Check FW version CPU / IO modules
4	14	1..10	31	31	44	Module has not passed factory test	Replace module
Error messages of the coupler interface							
Serious errors							
2	1..4/10	255	3	0	2	Same Node ID twice in the net	Use Node IDs only once
Light errors							
3	1..4/10	255	3	0	26	Incorrect or missing coupler configuration	Configure coupler
3	1..4/10	255	5	0..15	0..15	Error message of the operating system of the coupler:	see table "Coupler errors"

						Channel = Bit 4..7 Error = Bit 0..3	
3	1..4/10	255	6..12	0..15	0..15	Error message of the task x of the coupler: Task x = 'Module' - 5 Channel = Bit 4..7 Error = Bit 0..3	see table "Coupler errors"
3	1..4/10	255	31	0	33	Timeout while waiting for reset of the coupler	Check coupler
3	1..4/10	255	31	0	34	Timeout while waiting for readiness of the coupler	Check coupler
Warnings							
4	1..4/10	0..7	31	31	47	Short-circuit coupler/channel	Fix short-circuit
4	1..4/10	0..254	31	0..15	0..15	Communication error to the slave, Channel = Bit 4..7 Error = Bit 0..3	see table "Coupler errors"
4	1..4/10	255	2	0..15	0..15	Communication error of the coupler, Channel = Bit 4..7 Error = Bit 0..3	see table "Coupler errors"
Error messages of the serial interfaces							
Serious errors							
2	11..13	255	31	0	17	Access errorsr: - Interface could not be closed - Interface could not be opened - Timeslotmode could not be activated	Check FW version, replace CPU if necessary
Warnings							
4	13	255	4	0	42	Receiving error or timeout of the FBP slave interface	
Error messages of the CS31 bus (COM1 = CS31 master)							
Class	**Inter-face**	**Addr-ess**	**Module type**	**Chan-nel**	**Error identifier**	**Error message**	**Remedy**
Light errors							
3	11	0..61	0..5	0	8	No module found on the CS31 bus	Adapt configuration
3	11	0..61	1..8	0..31	0..63	S500 class 3 diagnostic sent by DC551	see tablee "S500 errors"
3	11	255	1	0	8	No module found on the CS31 bus	Check configuration
Warnings							
4	11	0..61	0..5	0	8	ICMK 14 with extensions configured, the extensions was not found on the bus	Check configuration
4	11	0..61	0..5	0	28	Module discarded and registered again; is only reported at the start	
4	11	0..61	0..5	0	32	Not configured module found on the bus, again discarded	Check CS31 bus, insert CS31_DIAG function block into the project
4	11	0..61	0..5	0..15	1	Internal error (error 1), reported by an AC31 I/O module	Check module
4	11	0..61	0..5	0..15	28	Configured module does not match the module registered to the bus	Check PLC configuration, insert CS31_DIAG function block into the project
4	11	0..61	0..5	0..15	32	Not configured module registered to the bus	Check PLC configuration, insert CS31_DIAG function block into the project
4	11	0..61	0..5	0..15	47	Short-circuit on CS31 module/channel	Fix short-circuit
4	11	0..61	0, 2, 4	0..15	2	Cut wire (Error 2) of an AC31 I/O module	Remove error at the module or the channel
4	11	0..61	0, 2, 4	0..15	4	Overload (Error 4) of an AC31 I/O module	
4	11	0..61	0, 2, 4	0..15	6	Overload + cut wire (Error 6) of an AC31 I/O module	
4	11	0..61	0, 2, 4	0..15	10	Short-circuit + cut wire or "out of range" at analog modules (Error 10) of an AC31 I/O module	
4	11	0..61	0, 2, 4	0..15	12	Overload + short-circuit (Error 12) of	

						an AC31 I/O module	
4	11	0..61	0, 2, 4	0..15	14	Short-circuit + overload + cut wire (Error 14) of an AC31 I/O module	
4	11	0..61	1, 3, 5	0..15	3	Analog value exceeded (Error 3) of an AC31 I/O module	
4	11	0..61	1, 3, 5	0..15	9	Cut wire (Error 9) of an AC31 analog module	
4	11	0..61	1..8	0..31	0..63	E4 error messages of DC551 and S500 I/O modules, see table below	see tablee "S500 errors"
4	11	0..61	31	31	9	Impossible configuration DC551 and S500 I/O modules (too many I/Os in one cluster)	Change configuration
4	11	0..61	31	31	31	Impossible configuration DC551 and S500 I/O modules (too many parameters in one cluster)	Change configuration
4	11	0..61	31	31	34	Outputs are written before the configuration of the modules DC551 + S500 I/O have been finished	

Notes

1) AC500 uses the following interface identification:
14 = I/O bus, 11 = COM1 (e.g., CS31 bus), 12 = COM2
FBP diagnosis blocks do not contain this identifier.

2) The assignment for "Device" is as follows:
31 = Module, 1..7 = Expansion 1..7

3) The assignment for "Module" is as follows:
31 = Module or module type (2=DO)

4) In case of module errors, "31 = Module" is reported for the channel.

5) In the current AC500 CPU firmware version, errors reported to the AC500 CPU by the DC505 FBP cannot be directly made visible on the display or in the PLC browser of the Control Builder PS501.

E1..E4	d1	d2	d3	d4	Identifier 000...063	AC500 display	<- displayed in 5)
Class	Comp	Dev	Mod	Ch	Err	PS501 PLC browser	
Byte 6 Bit 6..7	-	Byte 3	Byte 4	Byte 5	Byte 6 Bit 0..5	FBP diagnosis block	
Class	Inter-face	De-vice	Mod-ule	Chan-nel	Error identifier	Error message	Remedy
	1)	2)	3)	4)			
Errors of the S500 I/O modules							
S500 I/O module errors							
Light errors							
3	14	1..7	31	31	3	Timeout in the I/O module	Replace I/O module
	11 / 12	ADR	1..7				
3	14	1..7	31	31	9	Overflow diagnosis buffer	Restart
	11 / 12	ADR	1..7				
3	14	1..7	31	31	11	Process voltage too low	Check process voltage
	11 / 12	ADR	1..7				
3	14	1..7	31	31	19	Checksum error in the I/O module	Replace I/O module
	11 / 12	ADR	1..7				
3	14	1..7	31	31	26	Parameter error	Check master
	11 / 12	ADR	1..7				
3	14	1..7	31	31	36	Internal data interchange disturbed	Replace I/O module
	11 / 12	ADR	1..7				
3	14	1..7	31	31	40	Different hardware and firmware versions in the module	Replace I/O module
	11 / 12	ADR	1..7				
3	14	1..7	31	31	43	Interner Fehler im Gerät	Replace I/O module
	11 / 12	ADR	1..7				
3	14	1..7	31	31	47	Sensor voltage too low	Check sensor voltage
	11 / 12	ADR	1..7				
Warnings							
4	14	1..7	31	31	45	Process voltage switched off (ON->OFF)	Process voltage ON
	11 / 12	ADR	1..7				
Channel errors of the S500 I/O modules							
Warnings							
4	14	1..7	1	0..7	7	Measurement underflow at the analog input	Check input value
	11 / 12	ADR	1..7				
4	14	1..7	1	0..7	47	Short-circuit at the analog input	Check terminal
	11 / 12	ADR	1..7				
4	14	1..7	1	0..7	48	Measurement overflow or cut wire at the analog input	Check input value and terminal
	11 / 12	ADR	1..7				
4	14	1..7	2	0..23	47	Short-circuit at the digital output	Check terminal
	11 / 12	ADR	1..7				
4	14	1..7	3	0..7	48	Measurement overflow at the analog output	Check output value
	11 / 12	ADR	1..7				
4	14	1..7	3	0..7	7	Measurement underflow at the analog output	Check output value
	11 / 12	ADR	1..7				
Module errors DC551-CS31							
Light errors							
3	11	ADR	31	31	3	Timeout in the I/O module	Replace I/O module
3	11	ADR	31	31	9	Overflow of diagnosis buffer	Restart
3	11	ADR	31	31	11	Process voltage too low	Check process voltage
3	11	ADR	1..7	31	17	No communication with the I/O module	Replace I/O module
3	11	ADR	31	31	19	Checksum error in the I/O module	Replace I/O module
3	11	ADR	31	31	26	Parameter error	Check master
3	11	ADR	31	31	36	Internal data interchange disturbed	

3	11	ADR	31	31	40	Different hardware and firmware versions in the module	Replace I/O module
3	11	ADR	31	31	43	Internal error in the module	Replace I/O module
Warnings							
4	11	ADR	31	31	45	Process voltage ON/OFF	Process voltage ON
4	11	ADR	31/1..7	31	34	No response during initialization of the I/O module	Replace I/O module
4	11	ADR	31/1..7	31	32	Wrong I/O module on the slot	Replace I/O module and check configuration
Channel errors DC551-CS31							
Warnings							
4	11	ADR	31/1..7	8..23	47	Short-circuit at the digital output	Check terminal
Module errors DC505-FBP							
Light errors							
3	-	1..7	31	31	11	Process voltage too low	Check process voltage
3	-	1..7	31	31	17	No communication with the I/O module	Replace I/O module
3	-	31	31	31	3	Timeout in the I/O module	
3	-	31	31	31	19	Checksum error in the I/O module	
3	-	31	31	31	36	Internal data interchange disturbed	
3	-	31	31	31	40	Different hardware and firmware versions in the module	
3	-	31	31	31	43	Internal error in the module	
3	-	31	31	31	9	Overflow of diagnosis buffer	Restart
3	-	31	31	31	26	Parameter error	Check master
Warnings							
4	-	31	31	31	45	Process voltage ON/OFF	Process voltage ON
4	-	31/1..7	31	31	32	Wrong I/O module on the slot	Replace I/O module and check configuration
4	-	31/1..7	31	31	34	No response during initialization of the I/O module	Replace I/O module
Channel errors DC505-FBP							
Warnings							
4	-	31	2	8..15	47	Short-circuit at the digital output	Check terminal

Notes

1) AC500 uses the following interface identification:
14 = I/O bus, 11 = COM1 (e.g., CS31 bus), 12 = COM2
FBP diagnosis blocks do not contain this identifier.

2) The assignment for "Device" is as follows:
31 = Module, 1..7 = Expansion 1..7

3) The assignment for "Module" is as follows:
31 = Module or module type (2=DO)

4) In case of module errors, "31 = Module" is reported for the channel.

5) In the current AC500 CPU firmware version, errors reported to the AC500 CPU by the DC505 FBP cannot be directly made visible on the display or in the PLC browser of the Control Builder PS501.

5.2.5 Coupler errors

E1..E4	d1	d2	d3	d4	Identifier 000...063	AC500 display	
Class	Comp	Dev	Mod	Ch	Err	PS501 PLC browser	<- displayed in 5)
Byte 6 Bit 6..7	-	Byte 3	Byte 4	Byte 5	Byte 6 Bit 0..5	FBP diagnosis block	
Class	Interface	Device	Module	Channel	Error identifier	Error message	Remedy
	1)	2)	3)	4)			
Error of the coupler's operating system							
General operating system errors							
3	1..4/10	255	5	0	1	01$_{hex}$ = 1$_{dec}$ Error priority MAX	
3	1..4/10	255	5	0	2	02$_{hex}$ = 2$_{dec}$ Error priority NULL	
3	1..4/10	255	5	0	3	03$_{hex}$ = 3$_{dec}$ Error priority DOUBLE	
3	1..4/10	255	5	0	4	04$_{hex}$ = 4$_{dec}$ Stack size error	
3	1..4/10	255	5	0	5	05$_{hex}$ = 5$_{dec}$ EPROM size error	
3	1..4/10	255	5	0	6	06$_{hex}$ = 6$_{dec}$ RAM size error	
3	1..4/10	255	5	0	7	07$_{hex}$ = 7$_{dec}$ Segment counter error	
3	1..4/10	255	5	0	8	08$_{hex}$ = 8$_{dec}$ Segment size error	
3	1..4/10	255	5	0	9	09$_{hex}$ = 9$_{dec}$ Cycle time error	
3	1..4/10	255	5	0	10	0A$_{hex}$ = 10$_{dec}$ Frequency error	
3	1..4/10	255	5	0	11	0B$_{hex}$ = 11$_{dec}$ Trace buffer size error	
3	1..4/10	255	5	0	12	0C$_{hex}$ = 12$_{dec}$ Error min. RAM	
3	1..4/10	255	5	0	13	0D$_{hex}$ = 13$_{dec}$ Device address error	
3	1..4/10	255	5	0	14	0E$_{hex}$ = 14$_{dec}$ MCL token error	
3	1..4/10	255	5	0	15	0F$_{hex}$ = 15$_{dec}$ Driver type error	
3	1..4/10	255	5	1	0	10$_{hex}$ = 16$_{dec}$ SCC error	
3	1..4/10	255	5	1	1	11$_{hex}$ = 17$_{dec}$ Flash type OPT error	
3	1..4/10	255	5	1	2	12$_{hex}$ = 18$_{dec}$ Flash type BSL error	
3	1..4/10	255	5	1	3	13$_{hex}$ = 19$_{dec}$ Flash DIR name error	
3	1..4/10	255	5	1	4	14$_{hex}$ = 20$_{dec}$ Function table error	
3	1..4/10	255	5	1	5	15$_{hex}$ = 21$_{dec}$ RAM type error	
3	1..4/10	255	5	1	6	16$_{hex}$ = 22$_{dec}$ Flash DIR type error	
3	1..4/10	255	5	3	2	32$_{hex}$ = 50$_{dec}$ RAM test error	
3	1..4/10	255	5	3	3	33$_{hex}$ = 51$_{dec}$ Data segment error	
3	1..4/10	255	5	3	4	34$_{hex}$ = 52$_{dec}$ RAM error	
3	1..4/10	255	5	3	5	35$_{hex}$ = 53$_{dec}$ EPROM error	
3	1..4/10	255	5	3	6	36$_{hex}$ = 54$_{dec}$ DONGLE error	
3	1..4/10	255	5	3	7	37$_{hex}$ = 55$_{dec}$ Wrong RCS identifier error	
3	1..4/10	255	5	3	8	38$_{hex}$ = 56$_{dec}$ Error allocating memory	
3	1..4/10	255	5	6	4	64$_{hex}$ = 100$_{dec}$ RCS task not ready	
3	1..4/10	255	5	6	5	65$_{hex}$ = 101$_{dec}$ Task 1 not ready	
3	1..4/10	255	5	6	6	66$_{hex}$ = 102$_{dec}$ Task 2 not ready	
3	1..4/10	255	5	6	7	67$_{hex}$ = 103$_{dec}$ Task 3 not ready	
3	1..4/10	255	5	6	8	68$_{hex}$ = 104$_{dec}$ Task 4 not ready	
3	1..4/10	255	5	6	9	69$_{hex}$ = 105$_{dec}$ Task 5 not ready	
3	1..4/10	255	5	6	10	6A$_{hex}$ = 106$_{dec}$ Task 6 not ready	
3	1..4/10	255	5	6	11	6B$_{hex}$ = 107$_{dec}$ Task 7 not ready	
3	1..4/10	255	5	6	12	6C$_{hex}$ = 108$_{dec}$ Task 8 not ready	
3	1..4/10	255	5	6	13	6D$_{hex}$ = 109$_{dec}$ Task 9 not ready	
3	1..4/10	255	5	6	14	6E$_{hex}$ = 110$_{dec}$ Task 10 not ready	
3	1..4/10	255	5	6	15	6F$_{hex}$ = 111$_{dec}$ Task 11 not ready	
3	1..4/10	255	5	7	0	70$_{hex}$ = 112$_{dec}$ Task 12 not ready	
3	1..4/10	255	5	7	1	71$_{hex}$ = 113$_{dec}$ Task 13 not ready	

3	1..4/10	255	5	7	2	$72_{hex} = 114_{dec}$ Task 14 not ready	
3	1..4/10	255	5	7	3	$73_{hex} = 115_{dec}$ Task 15 not ready	
3	1..4/10	255	5	7	8	$78_{hex} = 120_{dec}$ MCL 0 missing	
3	1..4/10	255	5	7	9	$79_{hex} = 121_{dec}$ MCL 1 missing	
3	1..4/10	255	5	7	10	$7A_{hex} = 122_{dec}$ MCL 2 missing	
3	1..4/10	255	5	8	0	$80_{hex} = 128_{dec}$ MCL double	
3	1..4/10	255	5	8	1	$81_{hex} = 129_{dec}$ MCL start address	
3	1..4/10	255	5	8	2	$82_{hex} = 130_{dec}$ MCL 0 error	
3	1..4/10	255	5	8	3	$83_{hex} = 131_{dec}$ MCL 1 error	
3	1..4/10	255	5	8	4	$84_{hex} = 132_{dec}$ MCL 2 error	
3	1..4/10	255	5	8	10	$8A_{hex} = 138_{dec}$ MCL mode	
3	1..4/10	255	5	8	12	$8C_{hex} = 140_{dec}$ RCS 0 missing	
3	1..4/10	255	5	8	13	$8D_{hex} = 141_{dec}$ RCS 1 missing	
3	1..4/10	255	5	8	14	$8E_{hex} = 142_{dec}$ RCS 2 missing	
3	1..4/10	255	5	8	15	$8F_{hex} = 143_{dec}$ RCS 3 missing	
3	1..4/10	255	5	9	0	$90_{hex} = 144_{dec}$ RCS 4 missing	
3	1..4/10	255	5	9	1	$91_{hex} = 145_{dec}$ RCS 5 missing	
3	1..4/10	255	5	9	2	$92_{hex} = 146_{dec}$ RCS 6 missing	
3	1..4/10	255	5	9	3	$93_{hex} = 147_{dec}$ RCS 7 missing	
3	1..4/10	255	5	9	4	$94_{hex} = 148_{dec}$ RCS double	
3	1..4/10	255	5	9	5	$95_{hex} = 149_{dec}$ RCS start address	
3	1..4/10	255	5	9	6	$96_{hex} = 150_{dec}$ RCS 0 error	
3	1..4/10	255	5	9	7	$97_{hex} = 151_{dec}$ RCS 1 error	
3	1..4/10	255	5	9	8	$98_{hex} = 152_{dec}$ RCS 2 error	
3	1..4/10	255	5	9	9	$99_{hex} = 153_{dec}$ RCS 3 error	
3	1..4/10	255	5	9	10	$9A_{hex} = 154_{dec}$ RCS 4 error	
3	1..4/10	255	5	9	11	$9B_{hex} = 155_{dec}$ RCS 5 error	
3	1..4/10	255	5	9	12	$9C_{hex} = 156_{dec}$ RCS 6 error	
3	1..4/10	255	5	9	13	$9D_{hex} = 157_{dec}$ RCS 7 error	
3	1..4/10	255	5	10	0	$A0_{hex} = 160_{dec}$ LIB 0 missing	
3	1..4/10	255	5	10	1	$A1_{hex} = 161_{dec}$ LIB 1 missing	
3	1..4/10	255	5	10	2	$A2_{hex} = 162_{dec}$ LIB 2 missing	
3	1..4/10	255	5	10	3	$A3_{hex} = 163_{dec}$ LIB 3 missing	
3	1..4/10	255	5	10	4	$A4_{hex} = 164_{dec}$ LIB 4 missing	
3	1..4/10	255	5	10	5	$A5_{hex} = 165_{dec}$ LIB 5 missing	
3	1..4/10	255	5	10	6	$A6_{hex} = 166_{dec}$ LIB 6 missing	
3	1..4/10	255	5	10	7	$A7_{hex} = 167_{dec}$ LIB 7 missing	
3	1..4/10	255	5	10	8	$A8_{hex} = 168_{dec}$ LIB double	
3	1..4/10	255	5	10	9	$A9_{hex} = 169_{dec}$ LIB start address	
3	1..4/10	255	5	10	10	$AA_{hex} = 170_{dec}$ LIB 0 error	
3	1..4/10	255	5	10	11	$AB_{hex} = 171_{dec}$ LIB 1 error	
3	1..4/10	255	5	10	12	$AC_{hex} = 172_{dec}$ LIB 2 error	
3	1..4/10	255	5	10	13	$AD_{hex} = 173_{dec}$ LIB 3 error	
3	1..4/10	255	5	10	14	$AE_{hex} = 174_{dec}$ LIB 4 error	
3	1..4/10	255	5	10	15	$AF_{hex} = 175_{dec}$ LIB 5 error	
3	1..4/10	255	5	11	0	$B0_{hex} = 176_{dec}$ LIB 6 error	
3	1..4/10	255	5	11	1	$B1_{hex} = 177_{dec}$ LIB 7 error	
3	1..4/10	255	5	12	8	$C8_{hex} = 200_{dec}$ unknown IRQ	
3	1..4/10	255	5	12	9	$C9_{hex} = 201_{dec}$ Watchdog	
3	1..4/10	255	5	12	10	$CA_{hex} = 202_{dec}$ SCC TX IRQ	
3	1..4/10	255	5	12	11	$CB_{hex} = 203_{dec}$ SCC RX IRQ	
3	1..4/10	255	5	12	12	$CC_{hex} = 204_{dec}$ Task state	
3	1..4/10	255	5	14	6	$E6_{hex} = 230_{dec}$ Task 0	
3	1..4/10	255	5	14	7	$E7_{hex} = 231_{dec}$ Task 1	
3	1..4/10	255	5	14	8	$E8_{hex} = 232_{dec}$ Task 2	
3	1..4/10	255	5	14	9	$E9_{hex} = 233_{dec}$ Task 3	
3	1..4/10	255	5	14	10	$EA_{hex} = 234_{dec}$ Task 4	
3	1..4/10	255	5	14	11	$EB_{hex} = 235_{dec}$ Task 5	

3	1..4/10	255	5	14	12	EC$_{hex}$ = 236$_{dec}$ Task 6	
3	1..4/10	255	5	14	13	ED$_{hex}$ = 237$_{dec}$ Task 7	
3	1..4/10	255	5	15	0	F0$_{hex}$ = 240$_{dec}$ DBG task 0 segment	
3	1..4/10	255	5	15	1	F1$_{hex}$ = 241$_{dec}$ DBG task 1 segment	
3	1..4/10	255	5	15	2	F2$_{hex}$ = 242$_{dec}$ DBG task 2 segment	
3	1..4/10	255	5	15	3	F3$_{hex}$ = 243$_{dec}$ DBG task 3 segment	
3	1..4/10	255	5	15	4	F4$_{hex}$ = 244$_{dec}$ DBG task 4 segment	
3	1..4/10	255	5	15	5	F5$_{hex}$ = 245$_{dec}$ DBG task 5 segment	
3	1..4/10	255	5	15	6	F6$_{hex}$ = 246$_{dec}$ DBG task 6 segment	
3	1..4/10	255	5	15	7	F7$_{hex}$ = 247$_{dec}$ DBG task 7 segment	
General task errors							
3	1..4/10	255	6..12	0	1	01$_{hex}$ = 1$_{dec}$ No communication	
3	1..4/10	255	6..12	0	2	02$_{hex}$ = 2$_{dec}$ Idle	
3	1..4/10	255	6..12	3	2	32$_{hex}$ = 50$_{dec}$ Base initialization	
3	1..4/10	255	6..12	6	4	64$_{hex}$ = 100$_{dec}$ Parity error	
3	1..4/10	255	6..12	6	5	65$_{hex}$ = 101$_{dec}$ Frame error	
3	1..4/10	255	6..12	6	6	66$_{hex}$ = 102$_{dec}$ Overrun	
3	1..4/10	255	6..12	6	7	67$_{hex}$ = 103$_{dec}$ Data count	
3	1..4/10	255	6..12	6	8	68$_{hex}$ = 104$_{dec}$ Checksum error	
3	1..4/10	255	6..12	6	9	69$_{hex}$ = 105$_{dec}$ Timeout	
3	1..4/10	255	6..12	6	10	6A$_{hex}$ = 106$_{dec}$ Protocol error	
3	1..4/10	255	6..12	6	11	6B$_{hex}$ = 107$_{dec}$ Data error	
3	1..4/10	255	6..12	6	12	6C$_{hex}$ = 108$_{dec}$ NACK	
3	1..4/10	255	6..12	6	14	6E$_{hex}$ = 110$_{dec}$ Protocol base	
3	1..4/10	255	6..12	9	6	96$_{hex}$ = 150$_{dec}$ Invalid message header	
3	1..4/10	255	6..12	9	7	97$_{hex}$ = 151$_{dec}$ Invalid message length	
3	1..4/10	255	6..12	9	8	98$_{hex}$ = 152$_{dec}$ Invalid message command	
3	1..4/10	255	6..12	9	9	99$_{hex}$ = 153$_{dec}$ Invalid message structure	
3	1..4/10	255	6..12	9	10	9A$_{hex}$ = 154$_{dec}$ Message error	
3	1..4/10	255	6..12	9	11	9B$_{hex}$ = 155$_{dec}$ Message timeout	
3	1..4/10	255	6..12	9	12	9C$_{hex}$ = 156$_{dec}$ Invalid message sequence	
3	1..4/10	255	6..12	9	13	9D$_{hex}$ = 157$_{dec}$ Invalid message number	
3	1..4/10	255	6..12	9	14	9E$_{hex}$ = 158$_{dec}$ Unable to run the command, since execution of the previous command is not yet finished	
3	1..4/10	255	6..12	10	0	A0$_{hex}$ = 160$_{dec}$ Error in telegram header	
3	1..4/10	255	6..12	10	1	A1$_{hex}$ = 161$_{dec}$ Invalid device address	
3	1..4/10	255	6..12	10	2	A2$_{hex}$ = 162$_{dec}$ Wrong address data area	
3	1..4/10	255	6..12	10	3	A3$_{hex}$ = 163$_{dec}$ Data address and data count cause a buffer overflow	
3	1..4/10	255	6..12	10	4	A4$_{hex}$ = 164$_{dec}$ Invalid data index	
3	1..4/10	255	6..12	10	5	A5$_{hex}$ = 165$_{dec}$ Invalid data count	
3	1..4/10	255	6..12	10	6	A6$_{hex}$ = 166$_{dec}$ Unknown data type	
3	1..4/10	255	6..12	10	7	A7$_{hex}$ = 167$_{dec}$ Unknown function	
3	1..4/10	255	6..12	10	10	AA$_{hex}$ = 170$_{dec}$ Message base	
3	1..4/10	255	6..12	12	8	C8$_{hex}$ = 200$_{dec}$ Task not initialized, coupler not configured	
3	1..4/10	255	6..12	12	9	C9$_{hex}$ = 201$_{dec}$ Busy	
3	1..4/10	255	6..12	12	10	CA$_{hex}$ = 202$_{dec}$ No segment of RCS received	
3	1..4/10	255	6..12	12	11	CB$_{hex}$ = 203$_{dec}$ Unknown or wrong sender of a command message	
3	1..4/10	255	6..12	13	2	D2$_{hex}$ = 210$_{dec}$ No database	

3	1..4/10	255	6..12	13	3	D3hex = 211dec Error writing the database	
3	1..4/10	255	6..12	13	4	D4hex = 212dec Error reading the database	
3	1..4/10	255	6..12	13	5	D5hex = 213dec Error registering the diagnosis structure	
3	1..4/10	255	6..12	13	6	D6hex = 214dec Parameter error	
3	1..4/10	255	6..12	13	7	D7hex = 215dec Configuration	
3	1..4/10	255	6..12	13	8	D8hex = 216dec Function list	
3	1..4/10	255	6..12	13	9	D9hex = 217dec System	
3	1..4/10	255	6..12	13	10	DAhex = 218dec Not enough internal memory available	
3	1..4/10	255	6..12	13	11	DBhex = 219dec No DPR	
3	1..4/10	255	6..12	13	12	DChex = 220dec System base	

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**	**<- displayed in 5)**
Byte 6 Bit 6..7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0..5**	**FBP diagnosis block**	
Class	**Interface**	**Device**	**Module**	**Channel**	**Error identifier**	**Error message**	**Remedy**
	1)	**2)**	**3)**	**4)**			
Ethernet coupler errors							
IP task (task 7) errors, see remark 1							
3	1..4/10	255	12	3	2	32hex = 50dec No TCP task initialized	
3	1..4/10	255	12	3	3	33hex = 51dec Error when initializing the task configuration	
3	1..4/10	255	12	3	4	34hex = 52dec No Ethernet address	
3	1..4/10	255	12	3	5	35hex = 53dec Wait for warm start	
3	1..4/10	255	12	3	6	36hex = 54dec Invalid flags	
3	1..4/10	255	12	3	7	37hex = 55dec Invalid IP address	
3	1..4/10	255	12	3	8	38hex = 56dec Invalid net mask	
3	1..4/10	255	12	3	9	39hex = 57dec Invalid gateway	
3	1..4/10	255	12	3	11	3Bhex = 59dec Unknown hardware	
3	1..4/10	255	12	3	12	3Chex = 60dec No IP address	
3	1..4/10	255	12	3	13	3Dhex = 61dec Error initializing the driver	
3	1..4/10	255	12	3	14	3Ehex = 62dec No IP address configuration	
3	1..4/10	255	12	3	15	3Fhex = 63dec Invalid serial number	
3	1..4/10	255	12	4	0	40hex = 64dec No memory on chip	
3	1..4/10	255	12	6	14	6Ehex = 110dec Timeout	
3	1..4/10	255	12	6	15	6Fhex = 111dec Timeout invalid	
3	1..4/10	255	12	7	3	73hex = 115dec Target not reachable	
3	1..4/10	255	12	7	6	76hex = 118dec IP address invalid	
3	1..4/10	255	12	7	12	7Chex = 124dec Ethernet address invalid	
3	1..4/10	255	12	8	2	82hex = 130dec Unknown mode	
3	1..4/10	255	12	8	3	83hex = 131dec ARP cache full	
3	1..4/10	255	12	8	6	86hex = 134dec No ARP entry found	
3	1..4/10	255	12	9	5	95hex = 149dec Unexpected response	
TCP/UDP task (task 6) errors, see remark 1							
3	1..4/10	255	11	3	2	32hex = 50dec Init of IP task not completed	
3	1..4/10	255	11	3	3	33hex = 51dec Error when initializing the task configuration	

3	1..4/10	255	11	3	4	34$_{hex}$ = 52$_{dec}$ Init of IP task failed	
3	1..4/10	255	11	3	7	37$_{hex}$ = 55$_{dec}$ No memory available for init	
3	1..4/10	255	11	6	14	6E$_{hex}$ = 110$_{dec}$ Timeout	
3	1..4/10	255	11	6	15	6F$_{hex}$ = 111$_{dec}$ Invalid timeout	
3	1..4/10	255	11	7	0	70$_{hex}$ = 112$_{dec}$ Invalid socket	
3	1..4/10	255	11	7	1	71$_{hex}$ = 113$_{dec}$ Socket status	
3	1..4/10	255	11	7	3	73$_{hex}$ = 115$_{dec}$ Target not reachable	
3	1..4/10	255	11	7	4	74$_{hex}$ = 116$_{dec}$ Option not supported	
3	1..4/10	255	11	7	5	75$_{hex}$ = 117$_{dec}$ Invalid parameter	
3	1..4/10	255	11	7	6	76$_{hex}$ = 118$_{dec}$ Invalid IP address	
3	1..4/10	255	11	7	7	77$_{hex}$ = 119$_{dec}$ Invalid port	
3	1..4/10	255	11	7	8	78$_{hex}$ = 120$_{dec}$ CONN closed	
3	1..4/10	255	11	7	9	79$_{hex}$ = 121$_{dec}$ CONN reset	
3	1..4/10	255	11	7	10	7A$_{hex}$ = 122$_{dec}$ Unknown protocol	
3	1..4/10	255	11	7	11	7B$_{hex}$ = 123$_{dec}$ No sockets	
3	1..4/10	255	11	8	2	82$_{hex}$ = 130$_{dec}$ Unknown mode	
3	1..4/10	255	11	8	3	83$_{hex}$ = 131$_{dec}$ Max. data length exceeded	
3	1..4/10	255	11	8	4	84$_{hex}$ = 132$_{dec}$ Message count exceeded	
3	1..4/10	255	11	8	5	85$_{hex}$ = 133$_{dec}$ Max. group exceeded	
3	1..4/10	255	11	9	5	95$_{hex}$ = 149$_{dec}$ Unexpected response message	
OMB task (Modbus TCP) errors (task 3), see remark 1							
3	1..4/10	255	8	3	4	34$_{hex}$ = 52$_{dec}$ Invalid configuration data, server connections	
3	1..4/10	255	8	3	5	35$_{hex}$ = 53$_{dec}$ Invalid configuration data, task timeout	
3	1..4/10	255	8	3	6	36$_{hex}$ = 54$_{dec}$ Invalid configuration data, OMB timeout	
3	1..4/10	255	8	3	7	37$_{hex}$ = 55$_{dec}$ Invalid configuration data, mode	
3	1..4/10	255	8	3	8	38$_{hex}$ = 56$_{dec}$ Invalid configuration data, send timeout	
3	1..4/10	255	8	3	9	39$_{hex}$ = 57$_{dec}$ Invalid configuration data, connect timeout	
3	1..4/10	255	8	3	10	3A$_{hex}$ = 58$_{dec}$ Invalid configuration data, close timeout	
3	1..4/10	255	8	3	11	3B$_{hex}$ = 59$_{dec}$ Invalid configuration data, swap	
3	1..4/10	255	8	3	12	3C$_{hex}$ = 60$_{dec}$ TCP_UDP task not found, TCP task not ready	
3	1..4/10	255	8	3	13	3D$_{hex}$ = 61$_{dec}$ PLC task not found, PLC task not ready	
3	1..4/10	255	8	3	14	3E$_{hex}$ = 62$_{dec}$ Error initializing OMB task	
3	1..4/10	255	8	3	15	3F$_{hex}$ = 63$_{dec}$ Error initializing PLC task mode	
3	1..4/10	255	8	6	15	6F$_{hex}$ = 111$_{dec}$ Unknown sender of a response	
3	1..4/10	255	8	7	0	70$_{hex}$ = 112$_{dec}$ Error code in response	
3	1..4/10	255	8	7	1	71$_{hex}$ = 113$_{dec}$ No socket found in searched status	
3	1..4/10	255	8	7	2	72$_{hex}$ = 114$_{dec}$ Invalid value in request	
3	1..4/10	255	8	7	3	73$_{hex}$ = 115$_{dec}$ Error message of TCP task	
3	1..4/10	255	8	7	4	74$_{hex}$ = 116$_{dec}$ Modbus error	
3	1..4/10	255	8	7	5	75$_{hex}$ = 117$_{dec}$ No socket available	
3	1..4/10	255	8	7	6	76$_{hex}$ = 118$_{dec}$ Invalid socket handle	
3	1..4/10	255	8	7	7	77$_{hex}$ = 119$_{dec}$ Timeout in client socket	

3	1..4/10	255	8	7	8	$78_{hex} = 120_{dec}$ Socket closed, without response to command	
3	1..4/10	255	8	7	9	$79_{hex} = 121_{dec}$ Not ready flag set	
3	1..4/10	255	8	7	10	$7A_{hex} = 122_{dec}$ TCP task no longer ready	
3	1..4/10	255	8	7	11	$7B_{hex} = 123_{dec}$ Watchdog event	
3	1..4/10	255	8	7	12	$7C_{hex} = 124_{dec}$ Device in reconfiguration	
3	1..4/10	255	8	7	13	$7D_{hex} = 125_{dec}$ PLC task not initialized	
3	1..4/10	255	8	7	14	$7E_{hex} = 126_{dec}$ OMB server socket closed	

Remark 1:
The error information is also available at the output ERNO of the blocks used for the coupler.
The following applies: ERNO := 6000_{hex} OR error.

5.3 Diagnosis blocks for the AC500

The folder **"Diagnosis"** in the AC500 library **SysInt_AC500_Vxx.LIB** contains the following diagnosis blocks:

Block	Function
DIAG_EVENT	Generates an error entry in the diagnosis system
DIAG_GET	Provides detailed information and the error code for the next error of the selected error class
DIAG_INFO	Indicates that an error of the class 1..4 exists
DIAG_ACK	Acknowledges an error with error code
DIAG_ACK_ALL	Acknowledges all errors of an error class (except errors that have to be acknowledged exclusively)

The diagnosis blocks are described in detail in the documentation for the SysInt_AC500_Vxx.LIB library.

5.4 AC500-specific PLC browser commands

The PLC browser interface of the Control Builder provides CoDeSys standard commands as well as AC500-specific commands. The general operation of the PLC browser is described in the according user manual.

This section only describes AC500-specific commands and commands that provide special data for the AC500.

For all commands online help information is available. The help information is displayed language-dependent by entering "?command" when operating in online mode. The command "?" lists all available firmware commands.

The commands listed in online mode can differ from the commands shown when pressing the button [...] as the Control Builder version and firmware version can differ. The commands listed when clicking the button [...] are defined in the file "Browser.ini" that belongs to the selected target system package (TSP).

The PLC browser provides the following AC500-specific commands:

Command	Meaning	Implementation
?	Displays all implemented commands	Standard
mem	Memory dump from up to	Standard
memc	Memory dump relative to code area	Standard
memd	Memory dump relative to data area	Standard
reflect	Reflect actual command line (for test purposes)	Standard
dpt	Displays the data pointer table	Standard
ppt	Displays the block pointer table	Standard
pid	Displays the project ID	Standard

pinf	Displays project information in the format:
	`pinf`
	`Address of Structure: 16#0013CF74`
	`Date: 4213949F`
	`Project Name: MODBUS_Test_BB.pro`
	`Project Title: Test MODBUS`
	`Project Version: V1.0`
	`Project Author: Brigitte Blei`
	`Project Description:`
	`Test of serial interfaces`
	`End of Project-info.`
		Standard

tsk	Displays the IEC task list with task information in the format:
	`tsk`
	`Number of Tasks: 1`
	`Task 0: Main program, ID: 1519472`
	` Cycle count: 45402`
	` Cycle time: 1 ms`
	` Cycle time (min): 1 ms`
	` Cycle time (max): 1 ms`
	` Cycle time (avg): 1 ms`
	` Status: RUN`
	` Mode: CONTINUE`
	` ----`
	` Priority: 10`
	` Interval: 5 ms`
	` Event: NONE`
	` ----`
	` Function pointer: 16#00601584`
	` Function index: 131`
		Standard

startprg	Starts the user program	Standard
stopprg	Stops the user program	Standard
resetprg	Resets the user program	Standard
resetprgcold	Resets the user program (cold)	Standard
resetprgorg	Resets the user program (origin)	Standard
reload	Reloads the boot project from Flash	Standard

getprgprop	Displays program properties in the format:
	`getprgprop`
	`Name: MODBUS_FBP_Test_BB.pro`
	`Title: Test MODBUS`
	`Version: V1.0`
	`Author: Brigitte Blei`
	`Date: 4213949F`
		Standard

getprgstat	Displays the program status in the format:
	`getprgstat`
	`Status: Run`
	`Last error: Id 00000000 TimeStamp 000055F3`
	`Parameter 00000000 Text`
	`Flags:`
		Standard

filecopy	File command copy	No
filerename	File command rename	No
filedelete	File command delete	No
filedir	File command dir	No
saveretain	In V1.0 and V1.1: Saves the RETAIN variables to the SD card. As of V1.2: Writes the RETAIN variables to RAM (same as retain save)	Specific
restoreretain	In V1.0 and V1.1: Restores the RETAIN variables from the SD card. As of V1.2: Restores the RETAIN variables from RAM (same as retain restore)	Specific
setpwd	Sets the PLC password (required at logon!)	Standard
delpwd	Deletes the PLC password	Standard
plcload	Displays the PLC utilization (system+IEC+tasks+communication)	Standard

rtsinfo	Displays the firmware information (version, driver) in the format:
	rtsinfo
	`rts version: 2.4.5.2`
	`OS version: SMX smxPPC 3.5.2`
	`uses IO driver interface`
	`rts api version: 2.407`
	`4 driver(s) loaded`
	`driver 1: AC500 CPU driver, device interface version: 2.403`
	`driver 2: AC500 I/O-BUS driver, device interface version: 2.403`
	`driver 3: AC500 COM driver, device interface version: 2.403`
	`driver 4: AC500 Coupler driver, device interface version: 2.403`
	`AC500 PM581(DISP) : V1.0`
	`AC500 PM581(BOOT) : V1.2.0,(Build:May  3 2007,12:33:32,Release)`
	`AC500 PM581(FW)   : V1.2.0,(Build:May 10 2007,16:32:40,Release)`
		Specific

traceschedon	Enables task tracing	No
traceschedoff	Disables task tracing	No
traceschedstore	Stores task trace to RAM	No
ipaddr	Sets the IP address of the CPU	No
basetick	Sets the basetick to µs	No
diagreset	Resets the diagnosis system	Specific
diagack all	Acknowledges all errors (except errors that have to be quit exclusively)	Specific
diagack x	Acknowledges all errors of the class X (with X= 1...4)	Specific

diagshow all	Shows all errors in the format:
	`diagshow all`
	
	`--- All errors ---`
	`State             Class Comp Dev Mod Ch Err`
	
	`0152502216`
	`active and quitted  4    9    22  31 31  8`
	`       1970-01-01 00:00:08 occurred`
	`                           disappeared`
	`       1970-01-01 00:00:15 quitted`
		Specific

	`0152369165`
	`active not quitted  4    9    20  31  0 13`
	`       1970-01-01 01:19:12 occurred`
	`       -                   disappeared`
	`       -                   quitted`
	
	`--- end ---`	

time	Displays and sets the time of the realtime clock	Specific
date	Displays and sets the date of the realtime clock	Specific
batt	Polls the battery status	Specific
sdappl	Saves the boot project to the SD card	Specific
sdfunc	Displays and changes the SD card function	Specific
sdboot	Updates the bootcode from the SD card	Specific
sdfirm	Updates the firmware from the SD card	Specific
sdcoupler x	Updates the firmware of coupler x from the SD card	Specific
cpuload	Displays the CPU load (current, min., max., average)	Specific
delappl	Deletes the user program in the Flash memory	Specific
retain	Saving and restoring the RETAIN variables: retain clear -> Clears all RETAIN variables retain save -> Saves the RETAIN variables to the RAM disk retain restore -> Restores the RETAIN variables from the RAM disk retain export -> Exports the RETAIN variables from the RAM disk to the SD card retain import -> Imports the RETAIN variables from the SD card to the RAM disk	Specific as of V1.2
persistent	Saving and restoring the persistent area %R area: persistent clear -> Clears the %R area persistent save -> Saves the buffered %R area to the RAM disk persistent restore -> Restores the buffered %R area from the RAM disk persistent export -> Exports the buffered %R area from the RAM disk to the SD card persistent import -> Imports the buffered %R area from the SD card to the RAM disk	Specific as of V1.2
io-bus stat	Displays the I/O bus statistic	Specific
io-bus desc	Displays the I/O bus configuration	Specific
com protocols	Displays the protocols available for the serial interfaces	Specific
com settings	Displays the serial interface settings	Specific
coupler desc	Displays information on the coupler interfaces (type, firmware, serial number, date)	Specific
coupler settings	Displays the current coupler settings, for example, IP address and socket assignment	Specific as of V1.2
reboot	Reboots the PLC (CoDeSys performs a logout when restarting or logout possible up to 3 seconds after command input)	Specific

6 The SD memory card in the AC500

6.1 SD card functions

6.1.1 Summary of memory card functions

The AC500 controller contains a FLASH memory card of the type "SD Memory Card" (in short SD card) as external storage medium which is accessed by the PLC like a floppy disk drive. The SD card is used to transfer data between a commercially available PC with SD card interface and the AC500.

In the AC500, the SD card can be used to:

- update the AC500-CPU processor firmware
- update the CPU boot code
- update the display controller firmware (as of version V2.0)
- update the coupler firmware (as of version V1.2.0)
- update the firmware of the I/O modules connected to the I/O bus (as of version V2.0)
- load and save user programs (boot project)
- load and save the source code of the user program
- load and save retentive variables (RETAIN, %R area)
- load and save user data (with blocks)

The SD card can be operated by:

- writing/reading files using a standard PC card reader with SD card interface
- specific PLC browser commands
- reading and writing data from the user program using specific blocks

6.1.2 PLC browser commands for accessing the SD card

Command	Function
sdfunc	Displays and sets the SD card function "FunctionOfCard": 0 None 1 Load user program 2 Load firmware 3 Load user program and firmware
sdappl	Saves the user program (boot project) stored in Flash memory to the SD card (files Default.prg and Default.chk) and sets the SD card function to "Load user program" Set FunctionOfCard=+1 (bit 0 = 1)
saveretain	Up to V1.1: Saves the RETAIN variables "RETAIN.BIN" to the SD card (not from %M area)
restoreretain	Up to V1.1: Restores the RETAIN variables "RETAIN.BIN" from SD card to SRAM (not from %M area)
retain	As of V1.2: Saving and restoring the RETAIN variables: retain export -> Exports the RETAIN variables from the RAM disk to the SD card retain import -> Imports the RETAIN variables from the SD card to the RAM disk
persistent	As of V1.2: Saving and restoring the persistent area %R area: persistent export -> Exports the buffered %R area from the RAM disk to the SD card persistent import -> Imports the buffered %R area from the SD card to the RAM disk

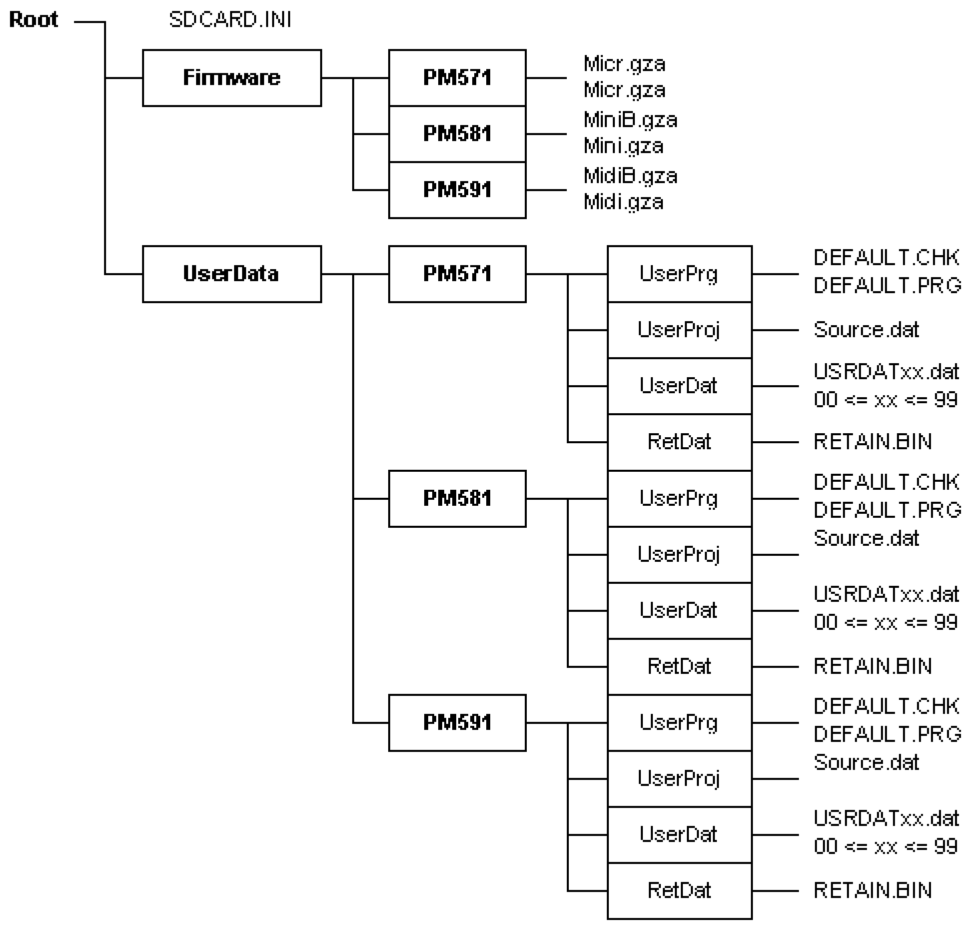
6.2 SD card file system

6.2.1 SD card file structure

In the PLC, the SD card is accessed like a PC floppy disk drive. The type of the file system is FAT (Microsoft DOS format). The file names are stored in 8.3 format (no "long" names) on the SD card.

File structure in versions V1.0 and V1.1

In versions V1.0 and V1.1, the file structure on the SD card looks as follows:



- The root directory on the SD card contains the command file SDCARD.INI (for a detailed description see the following section).

- The subdirectory "Firmware" contains further subdirectories for the:
  - CPUs PM571, PM581 and PM591.
  Each of these subdirectories contains the boot code MixxB.gza and the CPU firmware Mixx.gza. Boot code and firmware for the CPU PM582 are contained in the directory PM581.

- The subdirectory "UserData" contains the subdirectories:
  - "UserPrg", user program
  - "UserProj", source code of the user program
  - "UserDat", user data and
  - "RetDat", retentive data
  These directories contain the user data.

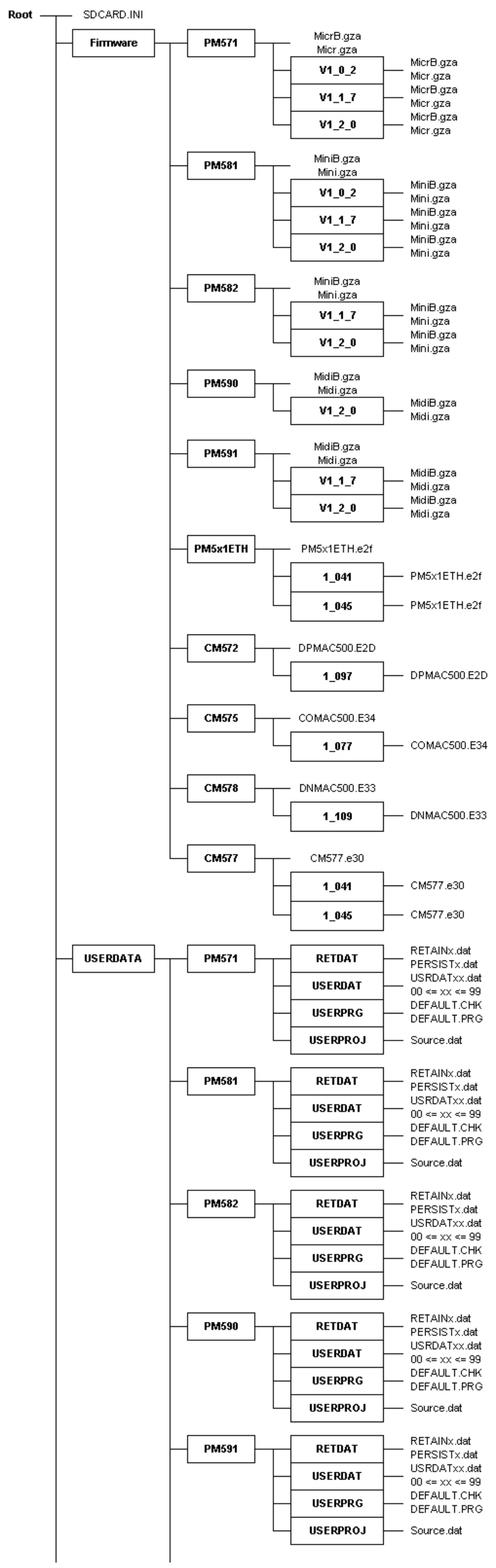
- xx in the file name represents the data file number.

File structure as of version V1.2

The SD card file structure has been revised and expanded for version V1.2. The following new functions are implemented:

1. Management of several boot code and firmware versions on one SD card.
2. Each CPU has its own directory.
   Up to version V1.1, there were only directories for controller classes PM57x, PM58x and PM59x. Due to this, the PM581 and PM582 data were stored to the same directory, for example ..\PM581.
3. Loading the field bus coupler firmware from the SD card by specific settings in the file SDCARD.INI or using the PLC browser command.
4. Management of several field bus coupler firmware versions.
5. Saving/restoring RETAIN data (%M area excluded) to/from the SD card via the user program and/or the PLC browser.
6. Saving/restoring the PERSISTENT area (%R area) to/from the SD card via the user program and/or the PLC browser.

As of version V1.2, the file structure looks as follows:


Root
SDCARD.INI
Firmware
PM571
MicrB.gza
Micr.gza
V1_0_2
V1_1_7
V1_2_0
MicrB.gza
Micr.gza
MicrB.gza
Micr.gza
MicrB.gza
Micr.gza
PM581
MiniB.gza
Mini.gza
V1_0_2
V1_1_7
V1_2_0
MiniB.gza
Mini.gza
MiniB.gza
Mini.gza
MiniB.gza
Mini.gza
PM582
MiniB.gza
Mini.gza
V1_1_7
V1_2_0
MiniB.gza
Mini.gza
MiniB.gza
Mini.gza
PM590
MidiB.gza
Midi.gza
V1_2_0
MidiB.gza
Midi.gza
PM591
MidiB.gza
Midi.gza
V1_1_7
V1_2_0
MidiB.gza
Midi.gza
MidiB.gza
Midi.gza
PM5x1ETH
PM5x1ETH.e2f
1_041
1_045
PM5x1ETH.e2f
PM5x1ETH.e2f
CM572
DPMAC500.E2D
1_097
DPMAC500.E2D
CM575
COMAC500.E34
1_077
COMAC500.E34
CM578
DNMAC500.E33
1_109
DNMAC500.E33
CM577
CM577.e30
1_041
1_045
CM577.e30
CM577.e30
USERDATA
PM571
RETDAT
USERDAT
USERPRG
USERPROJ
RETAINx.dat
PERSISTx.dat
USRDATxx.dat
00 <= xx <= 99
DEFAULT.CHK
DEFAULT.PRG
Source.dat
PM581
RETDAT
USERDAT
USERPRG
USERPROJ
RETAINx.dat
PERSISTx.dat
USRDATxx.dat
00 <= xx <= 99
DEFAULT.CHK
DEFAULT.PRG
Source.dat
PM582
RETDAT
USERDAT
USERPRG
USERPROJ
RETAINx.dat
PERSISTx.dat
USRDATxx.dat
00 <= xx <= 99
DEFAULT.CHK
DEFAULT.PRG
Source.dat
PM590
RETDAT
USERDAT
USERPRG
USERPROJ
RETAINx.dat
PERSISTx.dat
USRDATxx.dat
00 <= xx <= 99
DEFAULT.CHK
DEFAULT.PRG
Source.dat
PM591
RETDAT
USERDAT
USERPRG
USERPROJ
RETAINx.dat
PERSISTx.dat
USRDATxx.dat
00 <= xx <= 99
DEFAULT.CHK
DEFAULT.PRG
Source.dat

6.2.2 The command file "SDCARD.INI"

The command file "SDCARD.INI" located in the root directory on the SD card determines the behavior when starting the AC500 with installed SD card and when installing the card (firmware update, loading the user program, etc.).

The file SDCARD.INI is created in Windows INI format.

File content in versions V1.0 and V1.1

```
[Status]
FunctionOfCard=0

[FirmwareUpdate]
CPUPM5x1=0
CPUEC500=0
Display=0
Coupler_0=0
Coupler_1=0
Coupler_2=0
Coupler_3=0
Coupler_4=0

[UserProg]
UserProgram=0
RetainData=0
AddressData=0
```

The entries have the following meaning:

- **[Status]**
  FunctionOfCard=0

  The parameter FunctionOfCard determines which function is performed when inserting the SD card.
  With:
  FunctionOfCard=0 Perform no function when inserting the card or voltage ON
  FunctionOfCard=1 Load user program according to entry in group [UserProg]
  FunctionOfCard=2 Start firmware update according to entry in group [FirmwareUpdate]
  FunctionOfCard=4 Reserved for factory test

  If you want to load the firmware and user program, set FunctionOfCard = 3.

- **[FirmwareUpdate]**
  CPUPM5x1=0 (0 = no update, 1 = Update CPU firmware)
  Display=0 (0 = no update, 1 = Update display controller)
  Coupler_0=0 (0 = no update, 1 = Update internal coupler)
  Coupler_1=0 (0 = no update, 1 = Update coupler in slot 1)
  Coupler_2=0 (0 = no update, 1 = Update coupler in slot 2)
  Coupler_3=0 (0 = no update, 1 = Update coupler in slot 3)
  Coupler_4=0 (0 = no update, 1 = Update coupler in slot 4)

  The parameters in the group [FirmwareUpdate] specify which firmware is loaded. In version V1.0, only the CPU processor firmware can be loaded.

- **[UserProg]**
  UserProgram=0 (0 = no update, 1 = Update user program)
  RetainData=0 (0 = no update, 1 = Update RETAIN variables)

  The parameters in the group [UserProg] specify which user data are loaded. Up to version V1.1, only the user program can be loaded.

File content as of version V1.2

```
[Status]
FunctionOfCard=0

[FirmwareUpdate]
CPUPM5x1=0
CPUEC500=0
Display=0
Coupler_0=0
Coupler_1=0
Coupler_2=0
Coupler_3=0
Coupler_4=0

[UserProg]
UserProgram=0
RetainData=0
AddressData=0

[PM571]
TYPE=1
VERSION=1_2_0
PLCBOOT=1_2_0

[PM581]
TYPE=1
VERSION=1_2_0
PLCBOOT=1_2_0

[PM582]
TYPE=1
VERSION=1_2_0
PLCBOOT=1_2_0

[PM590]
TYPE=1
VERSION=1_2_0
PLCBOOT=1_2_0

[PM591]
TYPE=1
VERSION=1_2_0
PLCBOOT=1_2_0

[PM5x1]
TYPE=4
VERSION=1_045

[CM572]
TYPE=4
VERSION=1_097

[CM575]
TYPE=4
VERSION=1_077

[CM577]
TYPE=4
VERSION=1_045

[CM578]
TYPE=4
VERSION=1_109
```

The entries have the following meaning:

- **[Status]**
  FunctionOfCard=0

  The parameter FunctionOfCard determines which function is performed when inserting the SD card.
  With:

FunctionOfCard=0	Perform no function when inserting the card or voltage ON
FunctionOfCard=1	Load user program according to entry in group [UserProg]
FunctionOfCard=2	Update firmware according to entry in group [FirmwareUpdate]
FunctionOfCard=3	Update firmware according to entry in group [FirmwareUpdate] and load user program according to entry in [UserProg]
FunctionOfCard=4	Reserved for factory test
FunctionOfCard=8	Functions 0..4 + save debug data in case of possible failures

  Entry: 8 + function, Example: Function=1 ? FunctionOfCard=8+1=9

- **[FirmwareUpdate]**
  CPUPM5x1=0
  Display=0 (0 = no update, currently no further mode available)
  Coupler_0=0 (0 = no update, 2/3 = Update internal coupler)
  Coupler_1=0 (0 = no update, 2/3 = Update coupler in slot 1)
  Coupler_2=0 (0 = no update, 2/3 = Update coupler in slot 2)
  Coupler_3=0 (0 = no update, 2/3 = Update coupler in slot 3)
  Coupler_4=0 (0 = no update, 2/3 = Update coupler in slot 4)

  0 = no update,

  1 = Update CPU firmware from base directory on the CPU
  This mode is fully compatible to the firmware update of versions V1.0 and V1.1.

  2 = Update with specific version
  In mode 2, the update is only performed if the key 'version' in a product section (for example [PM581]) returns a different result than the version on the PLC. If the key 'version' is missing, no update is performed.

  3 = Update firmware only if the SD card firmware is newer than the PLC firmware.
  For mode 3 the same applies as for mode 2. However, an update is only performed if the firmware is newer than the PLC's firmware version.
  For this mode it must be ensured that the firmware versions can be read safely.

  The parameters of the group [FirmwareUpdate] specify which firmware is loaded.

- **[UserProg]**
  UserProgram=0 (0 = no update, 1 = Update user program)
  RetainData=0 (0 = no update, 1 = Update RETAIN variables)

  The parameters of the group [UserProg] specify which user data are loaded. In version V1.0, only the user program can be loaded.

- **[PM571] or [PM581], [PM582], [PM590], [PM591]**
  TYPE=1
  VERSION=1_2_0
  PLCBOOT=1_2_0

  Which firmware or boot code version is to be loaded is entered to the key for the according CPU PM5xy. The key is only evaluated for mode 2 and 3 in the key
  [FirmwareUpdate] / CPUPM5x1=2 or 3.

  The following applies for TYPE:
  TYPE=1 CPU
  TYPE=2 DISPLAY
  TYPE=3 I/O device at the CPU's I/O bus
  TYPE=4 Coupler (0-internal, 1..4-external)

The CPU firmware version is set with VERSION. For example, VERSION has to be set to VERSION=1_2_0 for version V1.2.0 and VERSION=1_1_7 for version V1.1.7.

The entry PLCBOOT specifies the boot code version of the CPU. For example, VERSION has to be set to VERSION=1_2_0 for version V1.2.0 and VERSION=1_1_3 for version V1.1.3.

- **[PM5x1] or [CM572], [CM575], [CM577], [CM578]**

  ```
  TYPE=4
  VERSION=1_045
  ```

  For the couplers always TYPE=4 has to be entered.

  VERSION specifies the firmware version of the coupler. For the Ethernet coupler, VERSION has to be set to, for example, VERSION=1_045 for version V01.045.

6.2.3 Initializing an SD card

6.2.3.1 Initializing an SD card using the AC500

The file structure described above is created on the SD card when a formatted SD card is inserted into the AC500. The file SDCARD.INI contains the following entries:

```
[Status]
FunctionOfCard=0
[FirmwareUpdate]
CPUPM5x1=0
Display=0
Coupler_0=0
Coupler_1=0
Coupler_2=0
Coupler_3=0
Coupler_4=0
[UserProg]
UserProgram=0
RetainData=0
AddressData=0
```

The following error message is displayed if you insert a non-formatted SD card:

```
152369164   FK4 Warning   Unable to read the SD card
```

Other files or subdirectories in the root directory on the SD card are kept unchanged.

6.2.3.2 Initializing an SD card using a PC

It is also possible to create the above described file structure on the hard disk of a PC with SD card interface.

To do this, create the required directories and an ASCII file named SDCARD.INI with Notepad, for example.

It is also possible to copy the file structure from an initialized SD card to the hard disk and then from the hard disk to SD cards that are not initialized.

6.3 Storing/loading the user program to/from an SD card

6.3.1 Storing the user program to an SD card

To store the user program to the SD card, proceed as follows:

1. Build the complete project using the menu items
   "Project" / "Clean all" and "Project" / "Rebuild all".

2. Load the project into the AC500.

3. Create the boot project on the controller using "Online" / "Create boot project".
   The boot project files (DEFAULT.PRG and DEFAUL.CHK) are loaded into the AC500 and flashed.
   The RUN LED on the AC500 flashes while data flashing is in progress.

4. Insert the SD card. If the SD card does not already contain the required file structure, the structure will be created
   (see also: “Initializing an SD card using the AC500”).

   ⚠ **Caution:** If a user program is already stored on the SD card, i.e., the directory UserData\PM5x1\UserPrg already contains the files DEFAULT.PRG and DEFAULT.CHK, these files will be overwritten without any warning.

   If you want to store several user programs to the SD card, you have to copy them into other directories using the PC.

5. Open the PLC Browser from the Resources tab and enter the command **"sdappl"<ENTER>**.
   The files DEFAULT.PRG and DEFAULT.CHK are loaded from the Flash memory and stored to the directory UserData\PM5x1\UserPrg on the SD card.
   In the file SDCARD.INI, the parameter "FunctionOfCard" is set to 1 (bit 0 = 1) and the parameter "UserProgram" is set to 1, i.e., the function "Load the user program" is activated.
   The RUN LED on the AC500 flashes while writing to the SD card is in progress.

If you insert the SD card containing the user program into the AC500, the SD card is loaded into the Flash memory of the AC500 (see next section).

6.3.2 Loading a user program from the SD card to the AC500

If an SD card is inserted into the AC500 when the **PLC is in STOP mode** or if the SD card is already inserted when switching on the control voltage, the file structure on the SD card is checked. If the file structure exists, the file SDCARD.INI is read. If the parameter "FunctionOfCard" is set to 1 (bit 0 = 1) and the parameter "UserProgram" = 1, the files DEFAULT.PRG and DEFAULT.CHK in the directory UserData\PM5x1\UserPrg on the SD card are loaded into the Flash memory of the AC500.

⚠ **Caution:** In versions V1.0 and V1.1, the user program can only be loaded with control voltage ON if FunctionOfCard=3 (i.e., firmware update and user program are loaded). To disable the firmware update, the according firmware file has to be deleted!

The RUN LED on the AC500 flashes while loading and flashing the user program is in progress.

The loaded program is activated after a PLC restart.

If the user program cannot be loaded (for example, due to missing files, wrong directory structure or mismatching project for the controller), a corresponding error message appears.

A summary of the SD card errors can be found in the section "SD card error messages".

If you insert the SD card into the AC500 when the **PLC is in RUN mode**, the user program is not loaded independent of the settings for the parameters "FunctionOfCard" (bit 0=1) and "UserProgram" (=1). Thus, the function "Load user program" can be deactivated with the PLC browser command "sdfunc 0" even if no PC card reader is available.

6.4 Storing/reading user data to/from an SD card

6.4.1 Structure of data files stored on the SD card

Depending on the AC500 CPU type, the data are stored in the following SD card directory:

AC500 CPU	Directory	File
PM571	..\UserData\PM571\UserDat	USRDATxx.DAT
PM581	..\UserData\PM581\UserDat	USRDATxx.DAT
PM591	..\UserData\PM591\UserDat	USRDATxx.DAT

A maximum of 100 files (USRDAT00.DAT...USRDAT99.DAT) can be stored in one directory.

Each data file USRDATxx.DAT can be divided into individual sectors, if necessary. The "sector label" enclosed in square brackets (such as [Sector_01]<CR><LF>) indicates the start of the sector. Within a sector, data are saved as data sets in ASCII format. The individual values of a data set are separated by semicolon. Each data set is closed with <CR><LF> ($0D_{hex}$, $0A_{hex}$).

This enables the direct import/export of the data from/to EXCEL. The data files can be viewed and edited using a standard ASCII editor (such as Notepad).

When saving/loading data files, observe the following rules:

- Data sets within a sector must always have the same number of values.
- Data sets in different sectors can have a different number of values.
- Values of integer data types can be stored. REAL or LREAL variables cannot be stored.
- The values of a data set must have the same data format (BYTE, WORD, INT,..).
- A sector can have data sets with different data format.
  (Warning: The user has to know the structure of the data for reading them.)
- The data sets are always appended to the end of the file when writing.
- Searching for a "sector label" within a file is possible when reading it.
- Data sets can be read starting from a particular "sector label".
- A particular data set of a sector cannot be read or written.
- If you want to read each data set individually, a "sector label" must be inserted before each data set.
- Reading and writing the data with help of the user program is done with the blocks SD_READ and SD_WRITE.
- The values of a data set must be available in variables successively stored in the PLC (e.g., ARRAY, STRING, %M area).
- A data file can be deleted with help of the PLC program.
- Individual data sets and/or sectors cannot be deleted with the user program.
  This has to be done on the PC using an ASCII editor such as Notepad.

Data file examples:

Example 1:

Data file USRDAT5.DAT without sectors:
-> 5 data sets, each with 10 DINT values:

```
600462;430;506;469;409;465;466;474;476;-1327203
600477;446;521;484;425;480;482;490;491;-1327187
600493;461;537;499;440;496;497;505;507;-1327172
600508;477;552;515;456;511;513;521;522;-1327156
600524;492;568;530;471;527;528;536;538;-1327141
```

Example 2:

Data file USRDAT7.DAT with sectors:
-> 3 sectors, each with 3 data sets and 10 DINT values per data set:

```
[Sector_01]
610439;10408;10483;10446;10387;10442;10444;10452;10453;-1317225
610455;10423;10499;10462;10402;10458;10460;10467;10469;-1317209
610476;10445;10520;10483;10424;10479;10481;10489;10490;-1317188

[Sector_02]
610570;10539;10614;10577;10518;10573;10575;10583;10584;-1317094
610585;10554;10630;10592;10533;10589;10591;10598;10600;-1317078
610602;10571;10646;10609;10550;10605;10607;10615;10616;-1317062

[Sector_03]
610701;10670;10746;10708;10649;10704;10706;10714;10715;-1316963
610717;10686;10761;10724;10665;10720;10722;10730;10731;-1316947
610739;10708;10783;10746;10686;10742;10744;10751;10753;-1316926
```

6.4.2 Blocks for storing/reading user data to/from the SD card

The following blocks are used to write and read user data from the PLC program to/from the SD card:

Library: SysInt_AC500_Vxx.lib

Folder: ..\Data Storage\ SD card
The library SysExt_AC500_Vxx.lib is also required. Both libraries are loaded automatically when creating a project for an AC500 CPU.

Blocks: SD_WRITE - Writes user data
SD_READ - Reads user data

The blocks SD_WRITE and SD_READ are described in the documentation for the block library SysInt_AC500_Vxx.lib.

The inputs and outputs of the block SD_WRITE and their functions are as follows:

Name	Type	Assignment
Inputs		
EN	BOOL	The FALSE->TRUE edge starts the write process
ATTRIB	BYTE	Write attribute of the block: 1 - Delete file 2 - Write append 3 - Write sector label
FILENO	BYTE	Consecutive file number 0 <= xx <= 99 (USRDATxx.DAT)
SEG	POINTER TO STRING	Pointer to sector label string (via ADR operator)
FORMAT	BYTE	Data format: 00 hex - 0 - BYTE 01 hex - 1 - CHAR 10 hex - 16 - WORD 11 hex - 17 - INT 20 hex - 32 - DWORD 21 hex - 33 - DINT
NVAR	WORD	Number of variables to be written
ADRVAR	DWORD	Address of the first variable, starting from which the data are available in the PLC (via ADR operator)
Outputs		
DONE	BOOL	Function completed
ERR	BOOL	Error: FALSE=no error, TRUE=error
ERNO	INT	Error number

The inputs and outputs of the block SD_READ and their functions are as follows:

Name	Type	Assignment
Inputs		
EN	BOOL	The FALSE->TRUE edge starts the read process
ATTRIB	BYTE	Read attribute of the block: 1 - Open file, seek sector, read data set 2 - Open file, read data set 3 - Open and read next data set 4 - Read data set, close file 5 - Close file
FILENO	BYTE	Consecutive file number 0 <= xx <= 99 (USRDATxx.DAT)
SEG	POINTER TO STRING	Pointer to sector label string (via ADR operator)
FORMAT	BYTE	Data format: 00 hex - 0 - BYTE 01 hex - 1 - CHAR 10 hex - 16 - WORD 11 hex - 17 - INT 20 hex - 32 - DWORD 21 hex - 33 - DINT
NVAR	WORD	Number of variables to be read
ADRVAR	DWORD	Address of the first variable starting from which the data are stored to the PLC (via ADR operator)
Outputs		
DONE	BOOL	Function completed
ERR	BOOL	Error: FALSE=no error, TRUE=error
ERNO	INT	Error number

The error messages of the blocks SD_READ and SD_WRITE are described in the chapter Function block error messages.

6.4.3 Deleting a data file stored on the SD card

To delete a data file from the SD card, proceed as follows:

1. Insert the SD card.

2. Call the block SD_WRITE with the following settings:

EN	:= TRUE	
ATTRIB	:= 1	(* delete *)
FILENO	:= 0...99	(* number of the file to be deleted *)

   SEG, FORMAT, NVAR, ADRVAR - any

6.4.4 Storing user data to the SD card - data file without sectors

Proceed as follows to store user data to the SD card in a data file without sectors:

1. Insert the SD card.

2. Write a data set by calling the block SD_WRITE with the following settings:

EN	:= TRUE	(* FALSE/TRUE edge starts writing *)
ATTRIB	:= 2	(* write append *)
FILENO	:= 0...99	(* number of the file to which the data set is to be appended *)
SEG	:= Address of the variable of the sector label (* any *)	
FORMAT	:= Data format	
NVAR	:= Number of values in data set	
ADRVAR	:= Address of the first variable to be written	

   If no corresponding file exists, then it is created.
   The write process is successfully completed when output DONE:=TRUE and output ERR:=FALSE. A write error is indicated by ERR:=TRUE and ERNO<>0.

3. Further data sets can be written with the same block settings after the completion message is indicated (output DONE=TRUE). This process is started with a FALSE/TRUE edge at input EN.

Note: The file USRDATxx.DAT is saved as USRDATxx.BAK for each write process and an "Open file / Write file / Close file" procedure is performed.

6.4.5 Storing user data to the SD card - data file with sectors

Proceed as follows to store user data to the SD card in a data file with sectors:

1. Insert the SD card.

2. Write the sector label by calling the block SD_WRITE with the following settings:

EN	:= TRUE	
ATTRIB	:= 3	(* write sector *)
FILENO	:= 0...99	(* number of the file to which the data set is to be appended *)
SEG	:= Address of the variable of the sector label	
FORMAT	:= Data format	
NVAR	:= Number of values in data set	
ADRVAR	:= Address of the first variable to be written	

   If no corresponding file exists, then it is created.
   The sector is successfully written when output DONE:=TRUE and output ERR:=FALSE. A write error is indicated by ERR:=TRUE and ERNO<>0.

3. Write a data set by calling the block SD_WRITE with the following settings:

EN	:= TRUE	(* FALSE/TRUE edge starts writing *)
ATTRIB	:= 2	(* write append *)
FILENO	:= 0...99	(* number of the file to which the data set is to be appended *)
SEG	:= Address of the variable of the sector label	
FORMAT	:= Data format	
NVAR	:= Number of values in data set	
ADRVAR	:= Address of the first variable to be written	

   The write process is successfully completed when output DONE:=TRUE and output ERR:=FALSE. A write error is indicated by ERR:=TRUE and ERNO<>0.

4. Further data sets can be written with the same block settings after the completion message is indicated (output DONE=TRUE). This process is started with a FALSE/TRUE edge at input EN.

5. If you want to write further sectors and data sets, repeat steps 2..4.

Note: The file USRDATxx.DAT is saved as USRDATxx.BAK for each write process and an "Open file / Write file / Close file" procedure is performed.

6.4.6 Loading user data from the SD card - data file without sectors

Proceed as follows to read user data from a data file without sectors on the SD card and write them to the PLC:

1. Insert the SD card.

2. Read a data set by calling the block SD_READ with the following settings:

EN	:= TRUE	(* FALSE/TRUE edge starts reading *)
ATTRIB	:= 2	(* open / read *)
FILENO	:= 0...99	(* number of the file which is to be read *)
SEG	:= Address of the variable of the sector label (* any *)	
FORMAT	:= Data format	
NVAR	:= Number of values in data set	
ADRVAR	:= Address of the first variable into which data are to be written	

   The read process is successfully completed when output DONE:=TRUE and output ERR:=FALSE. A read error is indicated by ERR:=TRUE and ERNO<>0.

3. Further data sets can be read with the following settings after the completion message is displayed (output DONE=TRUE). This process is started with a FALSE/TRUE edge at input EN:

EN	:= TRUE	(* FALSE/TRUE edge starts reading *)
ATTRIB	:= 3	(* continue read *)
FILENO	:= 0...99	(* number of the file which is to be read *)
SEG	:= Address of the variable of the sector label (* any *)	
FORMAT	:= Data format	
NVAR	:= Number of values in data set	
ADRVAR	:= Address of the first variable into which data are to be written	

   If an unexpected sector label or the end of file (EOF) is detected when reading, a corresponding error message is generated.

4. To read a further data set and to close the file afterwards, call the block SD_READ with the following settings after the completion message (output DONE=TRUE) and start the process with a FALSE/TRUE edge at input EN:

EN	:= TRUE	(* FALSE/TRUE edge starts reading *)
ATTRIB	:= 4	(* read / close *)
FILENO	:= 0...99	(* number of the file which is to be read *)
SEG	:= Address of the variable of the sector label (* any *)	
FORMAT	:= Data format	
NVAR	:= Number of values in data set	
ADRVAR	:= Address of the first variable into which data are to be written	

   If an unexpected sector label or the end of file (EOF) is detected when reading, a corresponding error message is generated.

5. To close the file, call the block SD_READ with the following settings after the completion message (output DONE=TRUE) and start the process with a FALSE/TRUE edge at input EN:

EN	:= TRUE	(* FALSE/TRUE edge starts close process *)
ATTRIB	:= 4	(* close *)
FILENO	:= 0...99	(* number of the file which is to be read *)
SEG	:= Address of the variable of the sector label (* any *)	
FORMAT	:= Data format	
NVAR	:= Number of values in data set (* any *)	
ADRVAR	:= Address of first variable (* any *)	

6.4.7 Loading user data from the SD card - data file with sectors

Proceed as follows to read user data from a data file with sectors on the SD card and write them to the PLC:

1. Insert the SD card.

2. Seek a sector label and read a data set by calling the block SD_READ with the following settings:
```
EN          := TRUE        (* FALSE/TRUE edge starts reading *)
ATTRIB      := 1           (* open / seek / read *)
FILENO      := 0...99      (* number of the file which is to be read *)
SEG         := Address of the variable of the sector label
FORMAT      := Data format
NVAR        := Number of values in data set
ADRVAR      := Address of the first variable into which data are to be written
```
   The read process is successfully completed when output DONE:=TRUE and output ERR:=FALSE. A seek error is indicated by ERR:=TRUE and ERNO<>0.

3. Further data sets can be read with the following settings after the completion message is indicated (output DONE=TRUE). This process is started with a FALSE/TRUE edge at input EN:
```
EN          := TRUE        (* FALSE/TRUE edge starts reading *)
ATTRIB      := 3           (* continue read *)
FILENO      := 0...99      (* number of the file which is to be read *)
SEG         := Address of the variable of the sector label (* any *)
FORMAT      := Data format
NVAR        := Number of values in data set
ADRVAR      := Address of the first variable into which data are to be written
```
   If an unexpected sector label or the end of file (EOF) is detected when reading, a corresponding error message is generated.

4. If you want to read further sectors / data sets, close the file and then repeat steps 2 and 3.

5. To read a further data set and to close the file afterwards, call the block SD_READ with the following settings after the completion message (output DONE=TRUE) and start the process with a FALSE/TRUE edge at input EN:
```
EN          := TRUE        (* FALSE/TRUE edge starts reading *)
ATTRIB      := 4           (* read / close *)
FILENO      := 0...99      (* number of the file which is to be read *)
SEG         := Address of the variable of the sector label
FORMAT      := Data format
NVAR        := Number of values in data set
ADRVAR      := Address of the first variable into which data are to be written
```
   If an unexpected sector label or the end of file (EOF) is detected when reading, a corresponding error message is generated.

6. To close the file, call the block SD_READ with the following settings after the completion message (output DONE=TRUE) and start the process with a FALSE/TRUE edge at input EN:
```
EN          := TRUE        (* FALSE/TRUE edge starts close process *)
ATTRIB      := 4           (* close *)
FILENO      := 0...99      (* number of the file to be read *)
SEG         := Address of the variable of the sector label
FORMAT      := Data format
NVAR        := Number of values in data set (* any *)
ADRVAR      := Address of first variable (* any *)
```

6.5 Storing and loading retentive data to/from an SD card

Retentive variables (RETAIN variables) declared with VAR_RETAIN .. END_VAR or VAR_GLOBAL RETAIN .. END_VAR are set to their initialization values during download. If the RETAIN variables shall keep their values also after a download, they have to be saved before starting the download and reloaded after the download is completed.

This is possible using the PLC browser commands "saveretain" and "restoreretain". The command "saveretain" saves the RETAIN variables to the file RETAIN.BIN on the SD card.

Depending on the CPU type, the files are stored to the following directories on the SD card:

AC500 CPU	Directory	Files
PM571	..\UserData\PM571\RetDat	RETAIN.BIN
PM581	..\UserData\PM581\RetDat	RETAIN.BIN
PM591	..\UserData\PM591\RetDat	RETAIN.BIN

6.6 Firmware update from the SD card

6.6.1 Storing the firmware to the SD card

Storing the firmware to the SD card is done using a standard PC card reader with SD card interface.

To do this, proceed as follows:

1. Initialize the SD card, i.e., create the file structure the PLC requires by, for example, inserting a new SD card into the AC500 (see also: "Initializing an SD card using the AC500").

2. Copy the firmware files into the corresponding directory:

AC500 CPU	Directory	File
PM571	..\Firmware\PM571	MICR.GZA
PM581	..\Firmware\PM581	MINI.GZA
PM591	..\Firmware\PM591	MIDI.GZA

3. Change the command file SDCARD.INI located in the root directory on the SD card as follows:
   Parameter "FunctionOfCard=2" (or =3 for firmware and user program update)
   Parameter CPUPM5x1=1

A specific firmware version can be loaded as of version V1.2. This is done by setting the parameter CPUPM5x1=2 or 3 and creating an according key for the CPU. The firmware has to be copied to the according directory. See the chapter The command file "SDCARD.INI".

6.6.2 Updating the firmware of the AC500 CPU from the SD card

To update the firmware of the AC500 CPU via SD card, proceed as follows:

1. Prepare the SD card as described in the section "Storing the firmware to the SD card".

2. Switch off the PLC control voltage.

3. Insert the SD card.

4. Switch on the PLC control voltage.
   The file structure on the SD card is checked when booting the PLC. If the file structure exists, the file SDCARD.INI is read. If the parameter "FunctionOfCard" is set to 2 (bit 1 = 1) and the parameter "CPUPM5x1" = 1, the file MIxx.gzaS19 is searched in the directory Firmware\PM5x1 (depends on the used CPU type) and then loaded into the PLC, checked and flashed.
   The individual steps are indicated as follows:

Process	Indication	Remark
Reading the firmware	RUN LED flashes fast	If you remove the SD card during reading, the previously stored firmware version is kept.
Flashing the firmware	RUN LED and ERR LED flash fast	**Warning:** If the control voltage is switched off during flashing, the firmware will be corrupted!
Firmware update completed successfully	RUN LED flashes slow (app. 1Hz)	
Incorrect firmware update	RUN LED and ERR LED flash slow (app. 1Hz)	

A specific firmware version can be loaded as of version V1.2. This is done by setting the parameter CPUPM5x1=2 or 3 and creating an according key for the CPU. The firmware has to be copied to the according directory. See the chapter The command file "SDCARD.INI".

Note: If the file SDCARD.INI contains the parameter setting "FunctionOfCard=3" (firmware update / load user program), first the firmware and then the user program are read from the SD card and then stored in the according Flash memory.

6.7 Writing and reading the project sources to/from the SD card

Usually it is sufficient to load the boot project (compiled user program) to the PLC. Sometimes, however, the user may wish to transfer the entire project sources to the PLC. This is why two different commands are available for this: The command "Online" => "Create boot project" writes the boot project to the flash memory of the PLC whereas the command "Online" => "Sourcecode download" stores the project sources to the SD card.

The project sources and all parts of a project are packed in the compressed file "SOURCE.DAT":

- All blocks in all IEC languages with all comments and symbols
- All visualizations
- Task configuration
- PLC configuration
- Online change information

The following is also contained if set accordingly in the project options:

- All loaded libraries
- All required configuration files

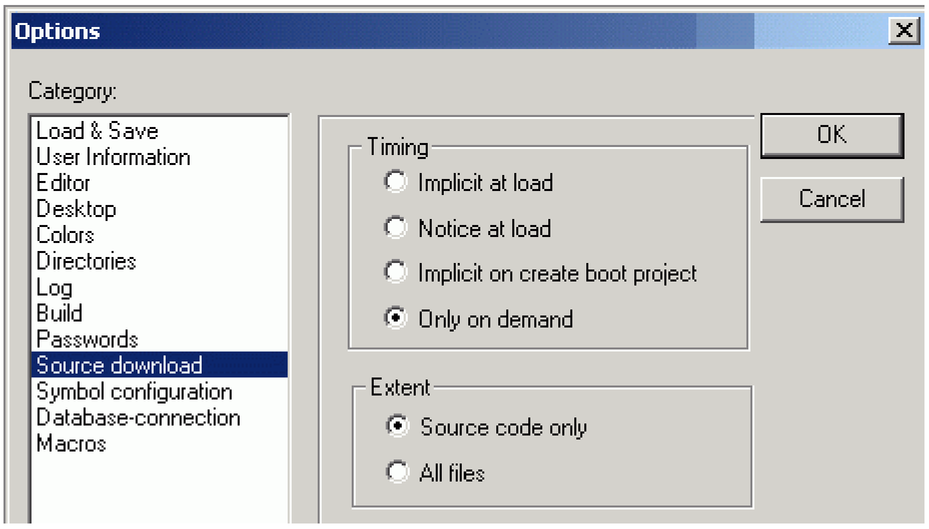
Reading the project sources can be done in two ways:

- Reading the compressed file SOURCE.DAT from the PLC (with an SD card inserted)
- Opening the compressed project directly on the PC using an SD card reader or opening a copy of the file SOURCE.DAT stored on the hard disk of the PC.

6.7.1 Writing the project sources from PC to SD card

Writing the project sources to the SD card is done as follows:

- Select the project sources to be written by selecting:
  "Project" => "Options" => "Source download"



Timing:
- Implicit at load
- Notice at load
- Implicit on create boot project
- [✓ Only on demand]

Extent:
- [✓ Source code only]
- All files

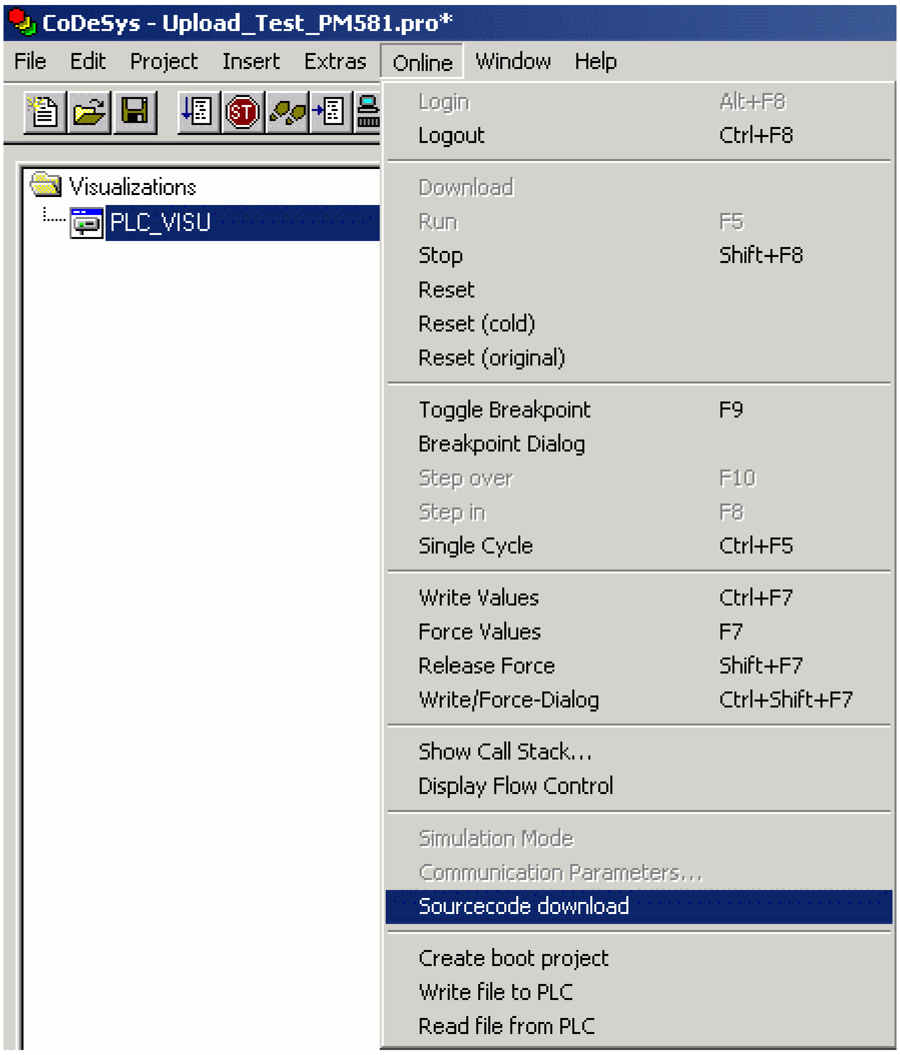
The "Timing" options specify the time when the project sources are to be written. The default setting is "On demand".
The "Extent" options define which files are to be written. Using the option "Source code only" writes all files that are part of the project. The option "All files" also includes the libraries and configuration files.
Especially with the option "All files" the compressed file SOURCE.DAT can have a size of several megabytes (in particular if visualizations with bitmaps are included!).

- Insert the SD card into the CPU. In case a new SD card is used, the directory structure required for the AC500 is created. This is indicated by the flashing LED RUN.

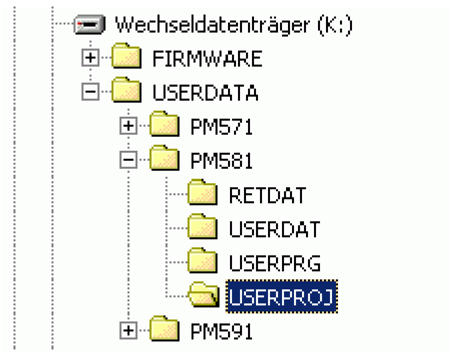
- Login to the PLC (via serial interface, Ethernet or ARCNET, depending on the interfaces available for your PLC).

- If possible, stop the PLC with the command "Online" => "Stop". (In Stop mode of the PLC, loading the project sources is much faster than in Run mode.)

- Select "Online" => "Sourcecode download".



- The compressed file SOURCE.DAT is created, downloaded to the PLC and written to the SD card. The file is located in the following directory on the SD card: [device]:\USERDATA\USERPROJ\SOURCE.DAT



- Restart the PLC with "Online" => "Start".

- Logout from the PLC by selecting "Online" => "Logout".

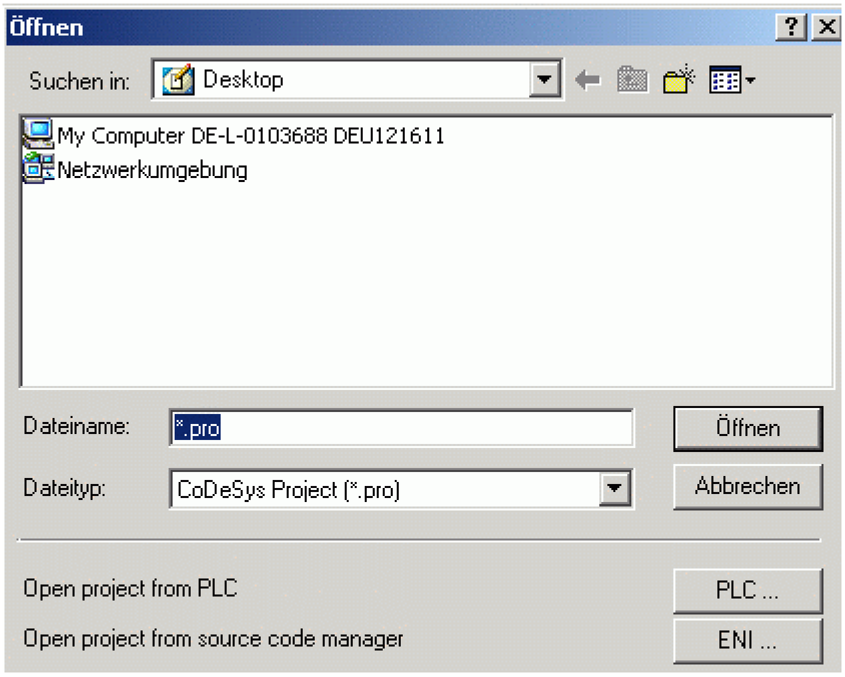
Example of the project sources size:

Project file	Upload_Test_PM581.pro	
Size of built user program	18638 Bytes	18 kB
Size of project sources (all files)	321503 Bytes	314 kB
Size of project sources (sources only)	90297 Bytes	89 kB

6.7.2 Loading the project sources from the PLC's SD card into the PC

Loading the project sources from the SD card into the PC is done as follows:

- Insert the SD card into the CPU (if not already inserted).
- Start the Control Builder PS501 (CoDeSys.exe).
- Select "File" => "Open".
- Click "PLC".


Öffnen

Suchen in: Desktop

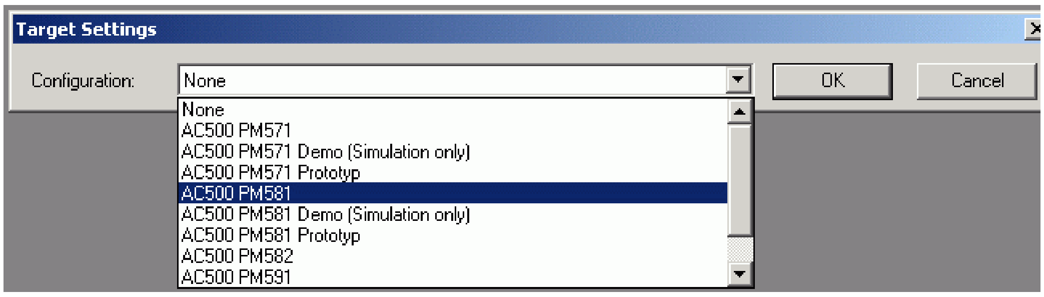
My Computer DE-L-0103688 DEU121611
Netzwerkumgebung

Dateiname: *.pro
Dateityp: CoDeSys Project (*.pro)

Öffnen
Abbrechen

Open project from PLC — PLC ...
Open project from source code manager — ENI ...

- Select the required CPU type (target), for example PM581.

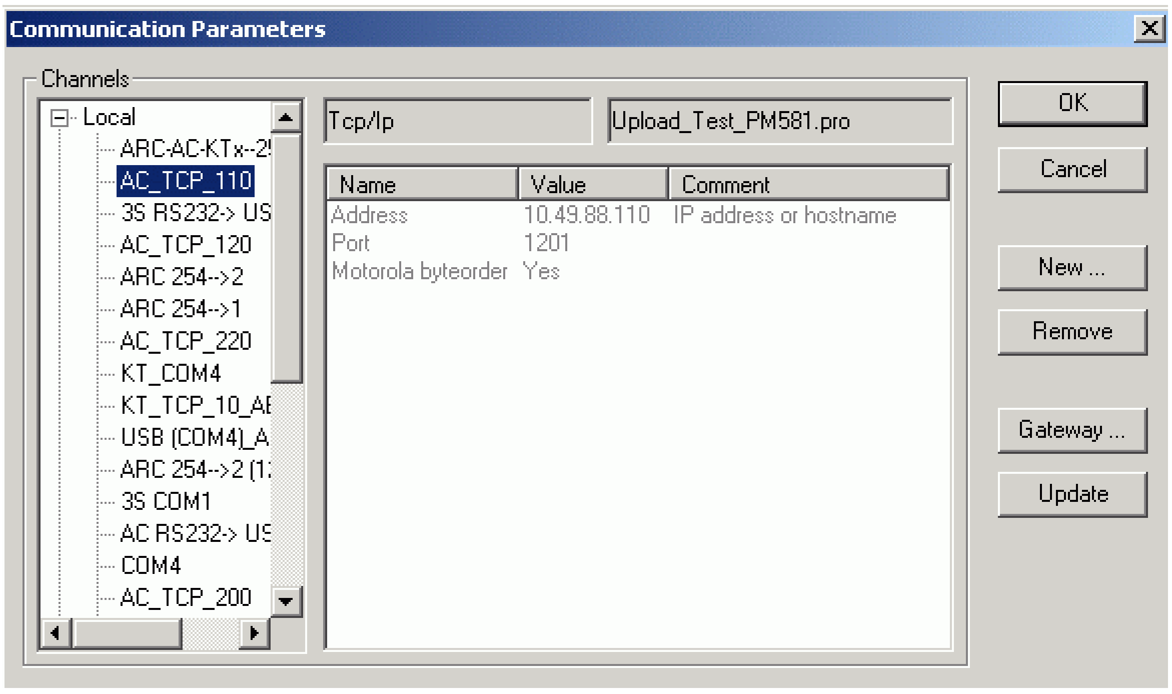

Target Settings

Configuration: None

OK
Cancel

None
AC500 PM571
AC500 PM571 Demo (Simulation only)
AC500 PM571 Prototyp
AC500 PM581
AC500 PM581 Demo (Simulation only)
AC500 PM581 Prototyp
AC500 PM582
AC500 PM591

- Select the communication channel, for example an Ethernet channel.


Communication Parameters

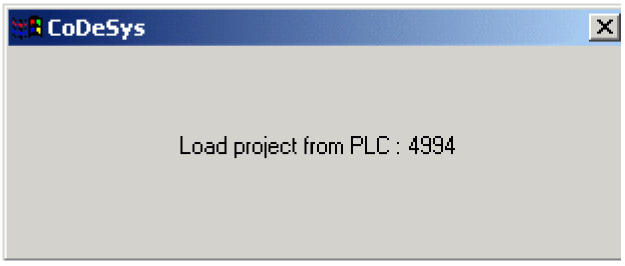
Channels

Local
- ARC-AC-KTx--2!
- AC_TCP_110
- 3S RS232-> US
- AC_TCP_120
- ARC 254-->2
- ARC 254-->1
- AC_TCP_220
- KT_COM4
- KT_TCP_10_AE
- USB (COM4)_A
- ARC 254-->2 (1:
- 3S COM1
- AC RS232-> US
- COM4
- AC_TCP_200

Tcp/Ip | Upload_Test_PM581.pro

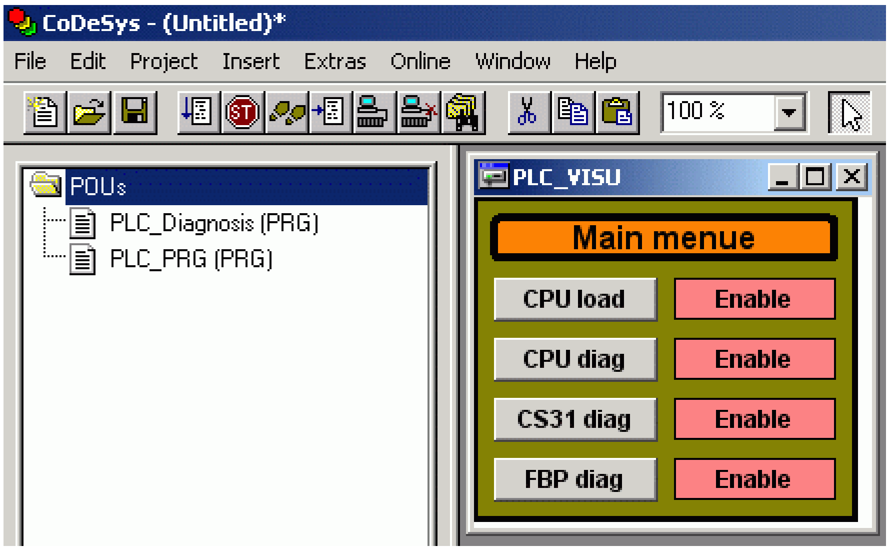
Name	Value	Comment
Address	10.49.88.110	IP address or hostname
Port	1201	
Motorola byteorder	Yes	

- The project sources are loaded block-wise.


CoDeSys

Load project from PLC : 4994

- Once the last block has been loaded, the project is decompressed by the software automatically. Now all project files are available.


CoDeSys - (Untitled)*

POUs
- PLC_Diagnosis (PRG)
- PLC_PRG (PRG)

PLC_VISU

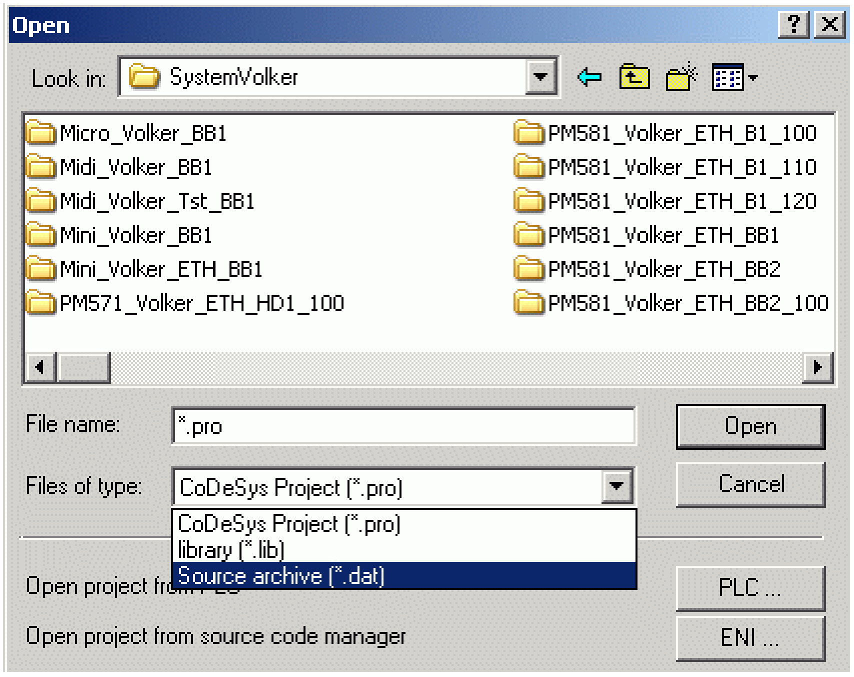
You can use this project to directly login to the PLC (no build required).

- If you want to change the project, save it under a new name, change and build the source code and then download the changed and built code (user program) to the PLC. Reload the boot project and the project sources.

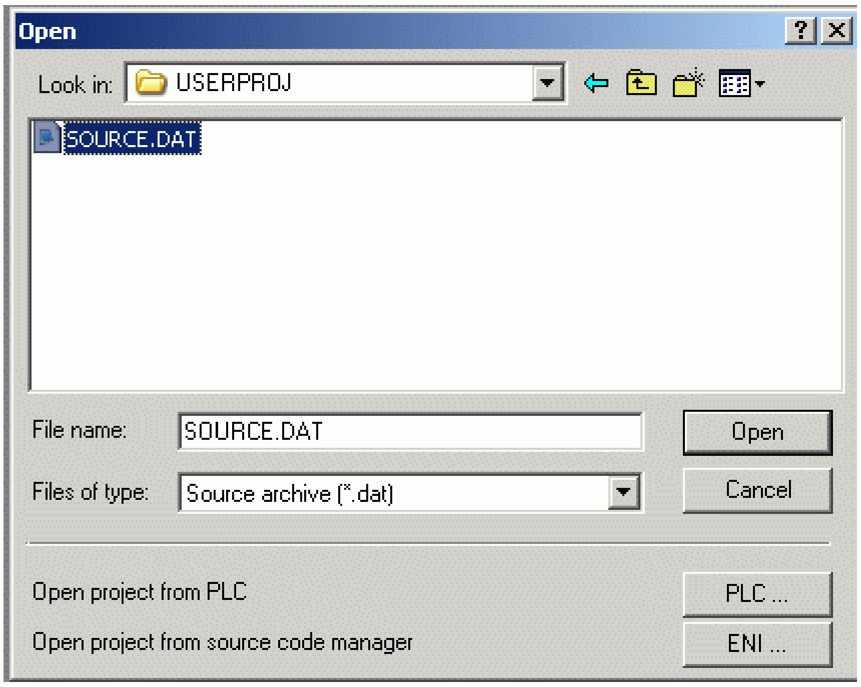
6.7.3 Loading the project sources from the SD card using the PC SD card reader

It is also possible to read the file SOURCE.DAT containing the project sources directly using the SD card reader for the PC. Proceed as follows:

- Insert the SD card into the PC SD card reader.
- Start the Control Builder PS501 (CoDeSys.exe).
- Select "File" => "Open".
- "Source archive (*.dat)" from the "Files of type" list box



- In the "Look in" list box, navigate to the SD card directory [Device]:\USERDATA\PM5x1\USERPROJ\SOURCE.DAT and select the file SOURCE.DAT. Click "Open".



- The project is loaded and then decompressed automatically.
  It is now possible to directly switch to online mode (without saving the project).

- If you want to change the project, save it under a new name, change and build it and then download the built user program with the boot project to the PLC. Insert the SD card into the PLC and rewrite the project sources to the SD card.

6.8 SD card error messages

The following table shows the error messages generated by the SD card.

Error number	Class	Description	Effect	Remedy
	E4 Warning	Incorrect directory structure when reading the user program	User program not loaded	Correct the directory structure with the PC
152369165	E4 Warning	Write error or SD card full	Data not written	Check the SD card with the PC and, if necessary, delete some files on the SD card or cut and paste them to the PC or insert another SD card
152369164	E4 Warning	Unable to read the SD card		Check the SD card with the PC
152369233	E4 Warning	Write-protection: Unable to initialize the SD card	Directory structure cannot be created	Remove write-protection
152369297	E4 Warning	Write-protection: Error writing the PLC program to the SD card	Data not written	Remove write-protection
152369361	E4 Warning	Write-protection: Error when storing the PLC firmware to the SD card	Data not written	Remove write-protection
152369425	E4 Warning	Write-protection: Error when storing the retentive variables (RETAIN.BIN) to the SD card	Data not written	Remove write-protection
152369489	E4 Warning	Write-protection: Error when writing the card function to the SD card	Data not written	Remove write-protection

7 Data storage in Flash memory

7.1 Blocks used for data storage

The library "SysInt_AC500_V10.lib" located in the directory "Data storage" / "Flash" contains the following blocks which are used to store data in the Flash memory:

Block	Function
FLASH_DEL	Deletes a data segment in the Flash memory
FLASH_READ	Reads a data segment from the Flash memory
FLASH_WRITE	Writes a data segment to the Flash memory

7.2 Example program for data storage

On the PS501 CD-ROM the directory:

[Device]:\CD_AC500\Examples\Flash_Data

contains the examples:

- FlashData_Structure_v10.pro Saves/loads structured data to/from Flash memory
- PM581_ST_Flash.pro Saves/loads data sets

8 Real-time clock and battery in the AC500

8.1 General notes concerning the real-time clock in the AC500

The real-time clock in the AC500 operates as a PC clock. It saves date and time to a DWORD in DT format (DATE AND TIME FORMAT), i.e., in seconds passed since the start time: 1 January 1970 at 00:00.

If a battery is connected and full, the real-time clock continues to run even if the control voltage is switched off.

If no battery is inserted or the battery is empty, the real-time clocks starts with the value 0 (=1970-01-01, 00:00:00).

When switching on the control voltage, the system clock of the operating system is set to the value of the real-time clock.

8.2 Setting and displaying the real-time clock

8.2.1 Setting and displaying the real-time clock with the PLC browser

The PLC browser commands **date** and **time** are used to set the real-time clock.

The commands date <ENTER> or time <ENTER> display the current date and time of the real-time clock.

The command:
date yyyy-mm-dd<ENTER> (year-month-day)
sets the date.

The command:
time hh-mm-ss<ENTER> (hours-minutes-seconds)
sets the time.

Example:
The real-time clock should be set to 22 February 2005, 16:50.

Enter the date:
date 2005-02-22<ENTER>

Display:
```
date 2005-02-22
Clock set to 2005-02-22 08:01:07
```
(Remark: the time remains unchanged)

Enter the time:
time 16:50<ENTER> (seconds are optional)

Display:
```
time 16:50
Clock set to 2005-02-22 16:50:00
```

8.2.2 Setting and displaying the real-time clock with the user program

The following blocks located in the folder "Realtime clock" of the system library SysExt_AC500_Vxx.LIB can be used to set and display the real-time clock (RTC) with help of the user program:

Block	Function
CLOCK	Sets and displays the real-time clock with values for year, month, day, hours, minutes and seconds. Also the day of week is indicated (Mo=0, Tue=1, Wed=2, Thu=3, Fr=4, Sa=5, Su=6). **Note:** The week of day cannot be set. It is given by the real-time clock. The input DAY_SET is ignored.
CLOCK_DT	Sets and displays the real-time clock in DT format, for example DT#2005-02-17-17:15:00.

The blocks CLOCK and CLOCK_DT are described in the documentation for the system library SysExt_AC500_Vxx.LIB.

8.3 The AC500 battery

The AC500 battery buffers the following data in case of "control voltage off":

- Retentive variables in SDRAM (VAR_RETAIN..END_VAR)
- File "Persistent.dat" in SDRAM (VAR_RETAIN PERSISTENT .. END_VAR) (in version V1.0.x only)
- Persistent data in %R area (as of version V1.2.0)
- Date and time of the real-time clock

If no battery is inserted or if the battery is empty, a warning (E4) is generated and the LED "ERR" lights up.

If no battery is required for the application (and thus no battery is inserted), a warning is generated and the error LED lights up each time the controller is switched on. To avoid this battery error indication, the parameter "Check Battery" is available under "CPU parameters" in the PLC configuration. The default setting of this parameter is "On", i.e., battery check is performed. If this parameter is set to "Off", the battery check is still performed and a corresponding error message is still generated each time the control voltage is switched on, but the system automatically quits this error and therefore the error LED does not light up (provided no further error exists).

The status of the battery can be checked in the PLC browser using the command "batt". The following is output:

0	Battery empty
20	Remaining battery charge below 20 %
100	Battery charge OK

In the user program, the battery status can be checked with the function "BATT" which is available in the folder "Battery" of the system library SysExt_AC500_Vxx.LIB. The following is output:

0	Battery empty
20	Remaining battery charge below 20 %, battery must be replaced
100	Battery charge OK

9 The fast counters in the AC500

9.1 Activating the fast counters via the I/O bus

The function "fast counters" is available in S500 I/O modules with firmware version V1.3 and later.

The digital I/O modules on the I/O bus contain two fast counters per module. If the I/O module does not have digital outputs, the corresponding counter modes are not valid. In case of an incorrect parameter setting, a diagnosis message is sent.

The fast counters are activated by setting the counter mode with the parameter "Fast counter" in the PLC configuration for the according I/O device (see also chapter "PLC configuration / I/O bus").

Control of the fast counter(s) is performed via the I/O data contained in the control byte of the submodule "Fast counter".

9.2 Counting modes of the fast counters

The counting modes of the fast counters are described in detail in the chapter "The fast counters of S500 I/O devices".

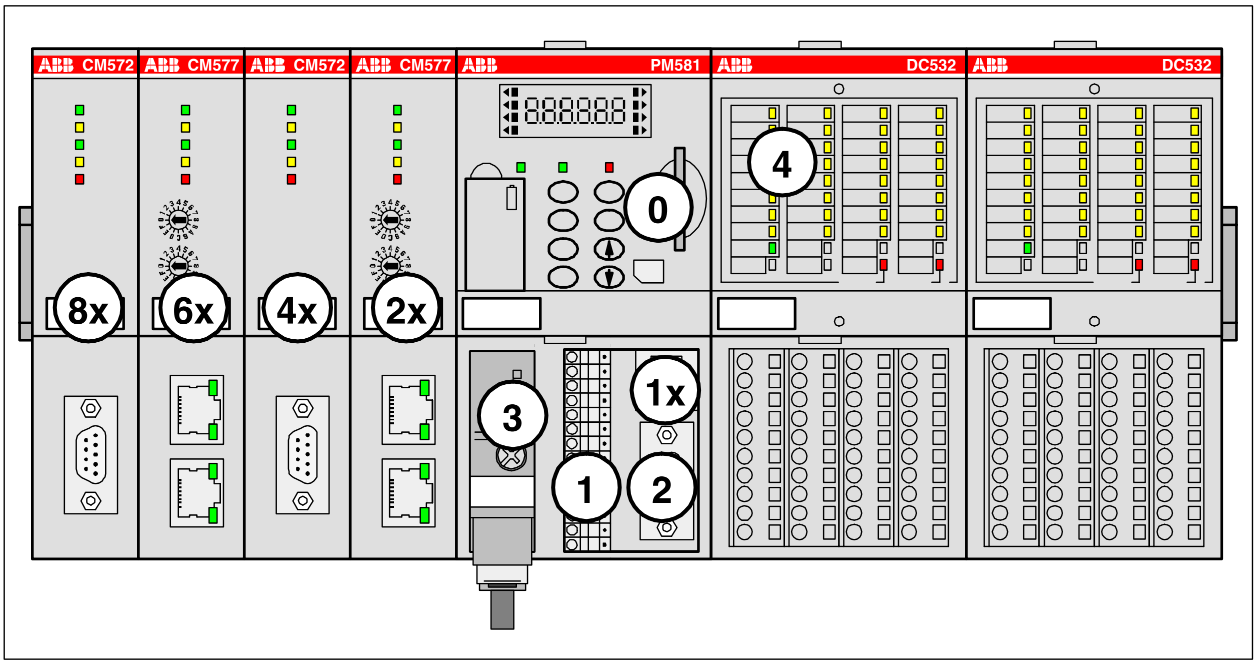
For an easy use, the blocks in the library Counter_AC500_V11.LIB can be used. These blocks are described in detail in the library documentation.

10 Programming and test

10.1 Programming interfaces to the AC500 used by the Control Builder

The AC500 controllers provide the following interfaces for communication with other devices:

No.	Designation	Interface	Programming access
0	CPU	Own CPU	CPU for online operation
1	COM1	Serial interface COM1	yes
2	COM2	Serial interface COM2	yes
3	FBP	FBP slave interface	yes (PM59x as of FW V1.2.0)
4	I/O bus	I/O bus	no
1x	Line 0	Internal coupler with channel 0 x 9	depends on type
2x	Line 1	Coupler inserted in slot 1 with chan. 0 x 19	depends on type
4x	Line 2	Coupler inserted in slot 2 with chan. 0 x 19	depends on type
6x	Line 3	Coupler inserted in slot 3 with chan. 0 x 19	depends on type
8x	Line 4	Coupler inserted in slot 4 with chan. 0 x 19	depends on type



Communication drivers:

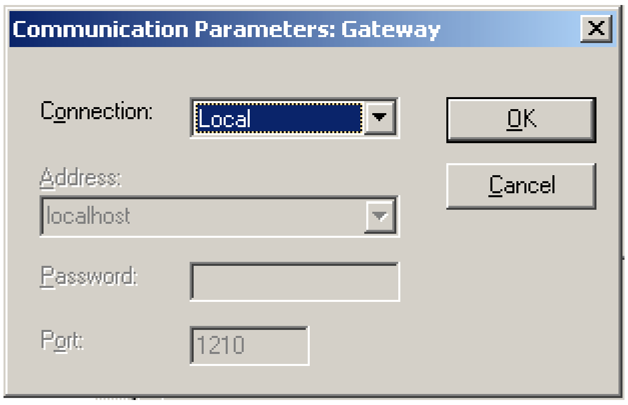
The following communication drivers are available for programming the AC500:

Serial (RS232)	Serial interface driver
ABB RS232 Route AC	Driver for serial interfaces with routing functionality (as of PS501 V1.2, routing as of firmware V1.3.x)
TCP/IP	Ethernet driver
ABB Tcp/Ip Level 2 AC	Ethernet driver with routing functionality (as of PS501 V1.2, routing as of firmware V1.3.x)
ABB Arcnet Route fast AC	Driver for programming via ARCNET with routing and adjustable block size (as of PS501 V1.2, routing as of firmware V1.3.x)
Serial (Modem)	Modem driver for modem connected to serial interface of the PC and PLC

Note: Routing is available with version V1.3.x of the Control Builder and PLC firmware.

Gateway configuration:

In the Control Builder, select "Online/Communication Parameters/Gateway" and select "Local" from the "Connection" list box (see also 3S Operation Manual / The Individual Components / Online Communication Parameters):


Communication Parameters: Gateway

Connection: Local

Address: localhost

Password:

Port: 1210

The following gateway settings apply for the Ethernet driver "ABB Tcp/Ip Level 2 AC":

Connection: Tcp/Ip
Address: localhost
Port: 1210

Setting the communication parameters:

The communication parameters and address data are set in the Control Builder by selecting the desired driver and specifying the parameters in the "Communication Parameters" window, which can be opened using the menu item "Online/Communication Parameters".

The following sections describe the drivers listed above and their settings.

10.2 Programming via the serial interfaces

The operation modes of the serial interfaces COM1 and COM2 are described in the chapter "PLC configuration". Both serial interfaces COM1 and COM2 are defined as programming interface by default.

The Installation guide provides information how to set the serial interfaces for the different PC operating systems.

PC and PLC are connected via the system cable TK501.

The interface parameters are set to fixed values and cannot be changed.

Baudrate: 19200 baud
Parity bit: no
Data bits: 8
Stop bits: 1
Synchronization: none
Motorola byteorder: yes

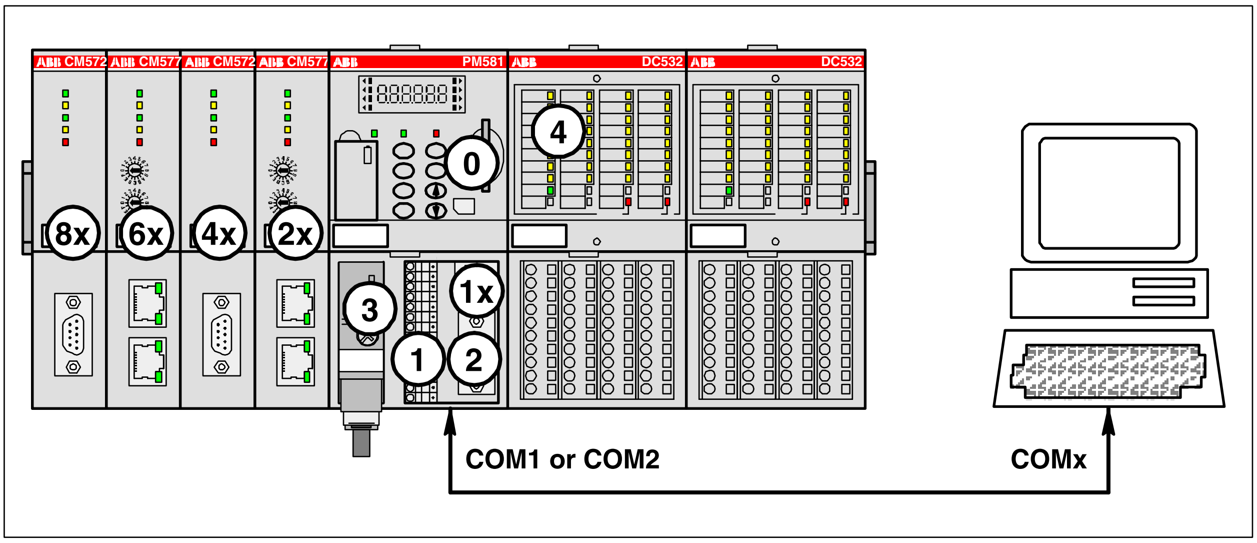
The following drivers are available for programming via the serial interfaces:

- Serial (RS232)
- ABB RS232 Route AC (as of PS501 V1.2, routing as of firmware V1.3.x)

10.2.1 Serial driver "Serial (RS232)"

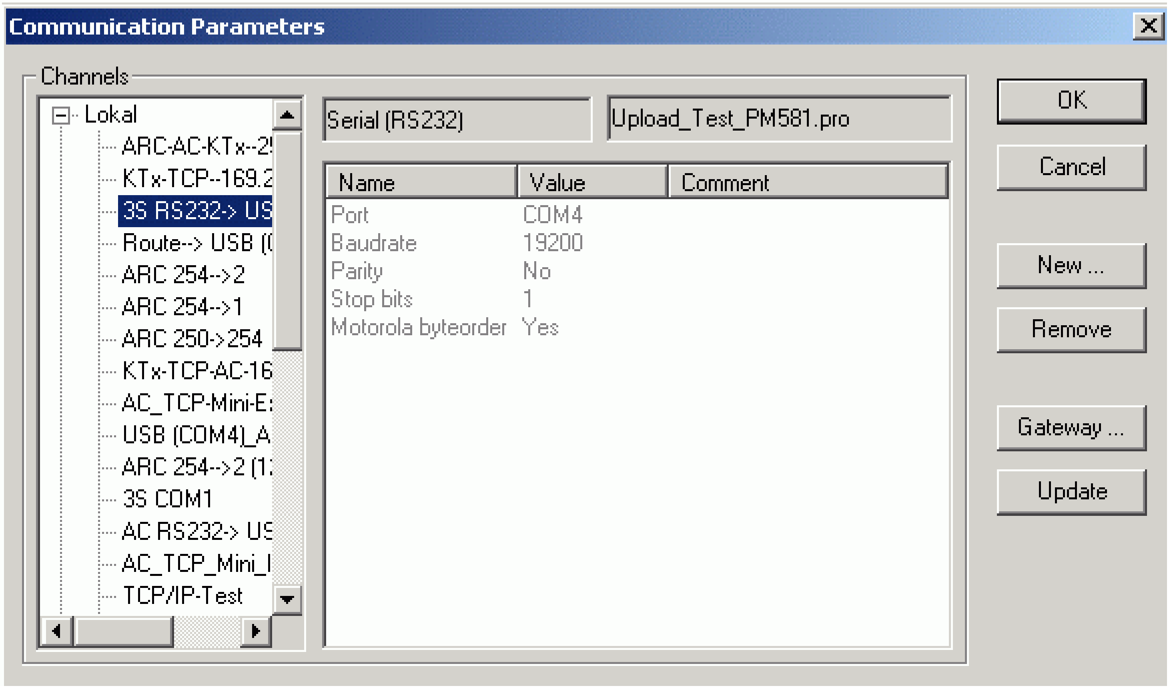
The serial driver "Serial (RS232)" provides the following functions:

- Online operation of the PLC with the Control Builder
- Online operation of the PLC with the fieldbus configurator SYCON.net
- OPC connection with OPC server, as of version V1.0
- Parallel operation of Control Builder and SYCON.net
- Parallel operation of Control Builder and OPC server



To define a new gateway channel for the serial driver, select "Online/Communication Parameters" and press the button "New" in the "Communication Parameters" window. In the appearing window, enter a name for the channel (for example USB->COM4) and select the driver "Serial RS232" from the device list. Select the desired values by **double clicking** the corresponding parameter:

Port: COM port on the PC
Baud rate: 19200 Baud
Motorola byteorder: Yes


Communication Parameters

Channels
- Lokal
  - ARC-AC-KTx--2!
  - KTx-TCP--169.2
  - 3S RS232-> US
  - Route--> USB (I
  - ARC 254-->2
  - ARC 254-->1
  - ARC 250->254
  - KTx-TCP-AC-16
  - AC_TCP-Mini-E:
  - USB (COM4)_A
  - ARC 254-->2 (1:
  - 3S COM1
  - AC RS232-> US
  - AC_TCP_Mini_I
  - TCP/IP-Test

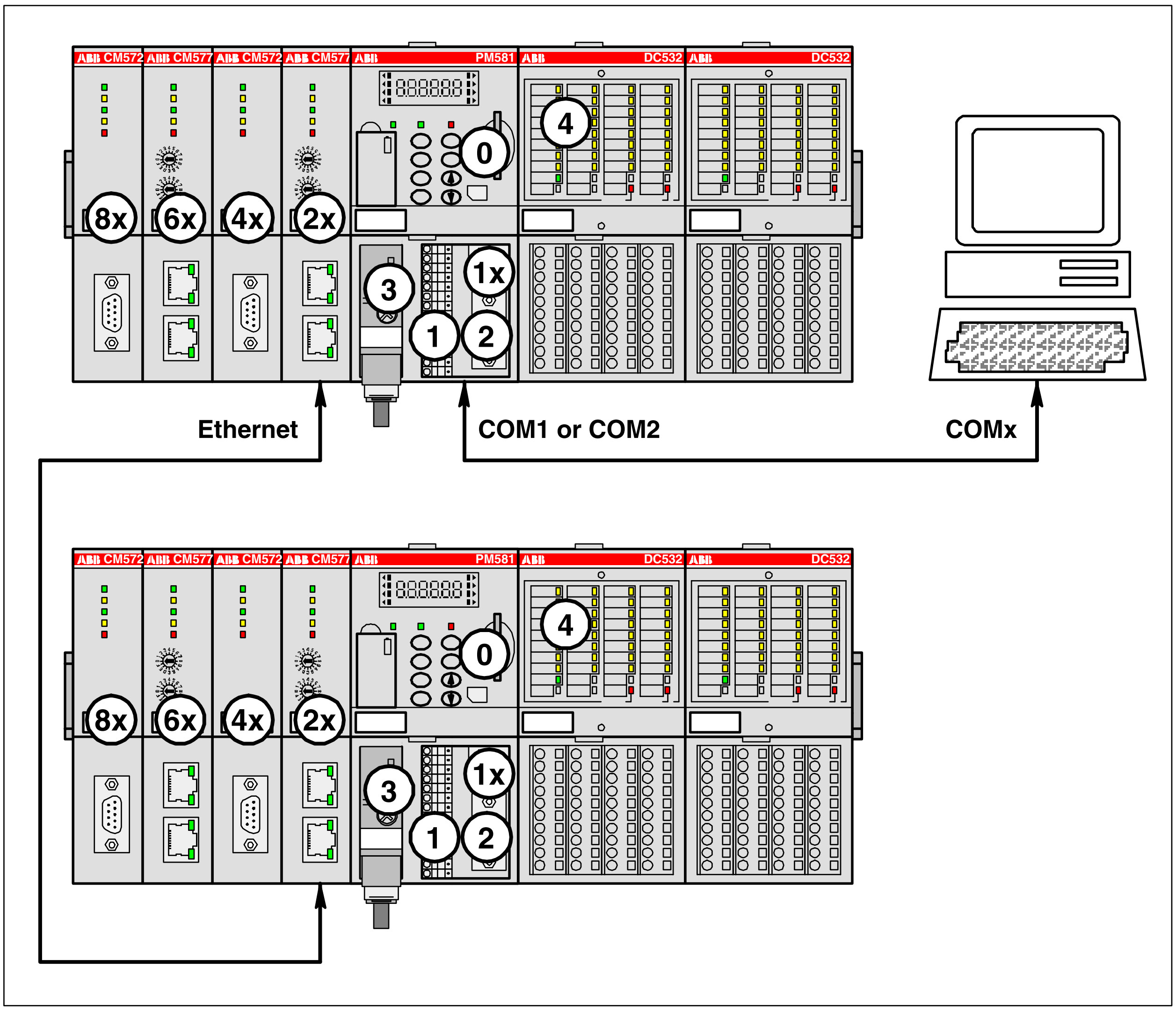
Serial (RS232) | Upload_Test_PM581.pro

Name	Value	Comment
Port	COM4	
Baudrate	19200	
Parity	No	
Stop bits	1	
Motorola byteorder	Yes	

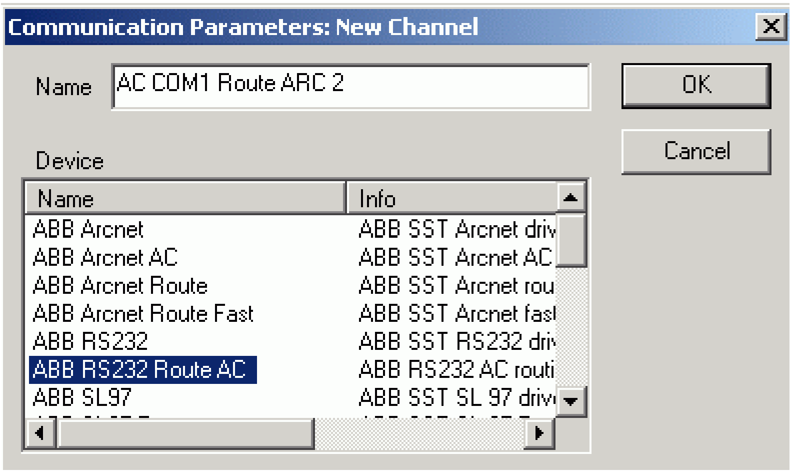
10.2.2 Serial driver "ABB RS232 Route AC"

As of version V1.2.0, the serial routing driver "ABB RS232 Route AC" is available in addition to the serial driver. This driver provides the following functions:

- Online operation of the PLC with the Control Builder
- Online operation of the PLC with the fieldbus configurator SYCON.net
- OPC connection with OPC server, as of version V1.2.0
- Online operation of PLCs connected via Ethernet or ARCNET using the serial interface (Control Builder version V1.3 and later)
- Parallel operation of Control Builder and SYCON.net
- Parallel operation of Control Builder and OPC server
- Online operation of AC31 series 90 controllers (07KT9x)



To define a new gateway channel for the serial routing driver, select "Online/Communication Parameters" and press the button "New" in the "Communication Parameters" window. In the appearing window, enter a name for the channel (for example AC COM1 Route ARC 2) and select the driver "ABB RS232 Route AC" from the device list.



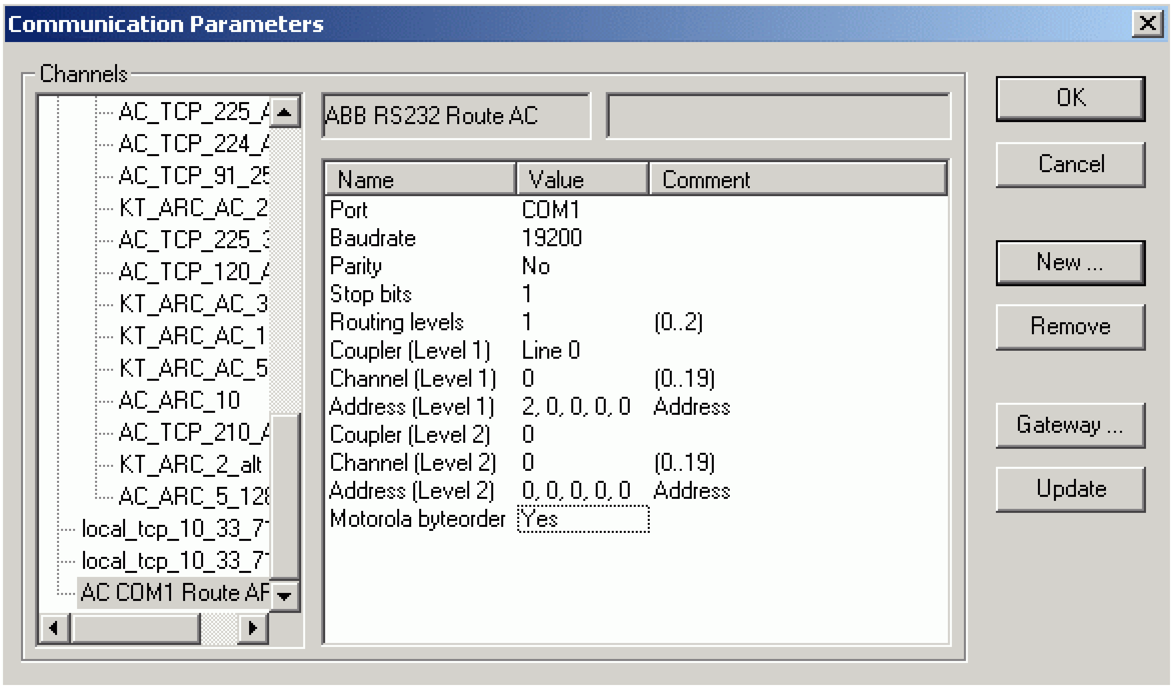
Communication Parameters: New Channel

Name: AC COM1 Route ARC 2

Device

Name	Info
ABB Arcnet	ABB SST Arcnet driv
ABB Arcnet AC	ABB SST Arcnet AC
ABB Arcnet Route	ABB SST Arcnet rou
ABB Arcnet Route Fast	ABB SST Arcnet fast
ABB RS232	ABB SST RS232 driv
ABB RS232 Route AC	ABB RS232 AC routi
ABB SL97	ABB SST SL 97 driv

The following communication parameters can be set for the serial routing driver "ABB RS232 Route AC":



Communication Parameters

Channels: AC_TCP_225_A, AC_TCP_224_A, AC_TCP_91_25, KT_ARC_AC_2, AC_TCP_225_3, AC_TCP_120_A, KT_ARC_AC_3, KT_ARC_AC_1, KT_ARC_AC_5, AC_ARC_10, AC_TCP_210_A, KT_ARC_2_alt, AC_ARC_5_128, local_tcp_10_33_7, local_tcp_10_33_7, AC COM1 Route AR

ABB RS232 Route AC

Name	Value	Comment
Port	COM1	
Baudrate	19200	
Parity	No	
Stop bits	1	
Routing levels	1	(0..2)
Coupler (Level 1)	Line 0	
Channel (Level 1)	0	(0..19)
Address (Level 1)	2, 0, 0, 0, 0	Address
Coupler (Level 2)	0	
Channel (Level 2)	0	(0..19)
Address (Level 2)	0, 0, 0, 0, 0	Address
Motorola byteorder	Yes	

Parameter	Possible values	Meaning
Port	COMx (PC dependant)	Serial interface of the PC
Baudrate	19200	Always 19200 baud
Parity	No	Always no parity
Stop bits	1	Always one stop bit
Routing levels	0...2	Routing levels (0 = none)
Coupler (Level 1)	0, line 0...line 4	Coupler for level 1
Channel (Level 1)	0...19	Channel on coupler level 1
Address (Level 1)	0, 0, 0, 0, 0 (max. 5 bytes)	Address in target coupler level 1
Coupler (Level 2)	0, line 0...line 4	Coupler for level 2
Channel (Level 2)	0...19	Channel on coupler level 2
Address (Level 2)	0, 0, 0, 0, 0 (max. 5 bytes)	Address in target coupler level 2
Motorola byteorder	yes	Selection of Motorola or Intel byteorder

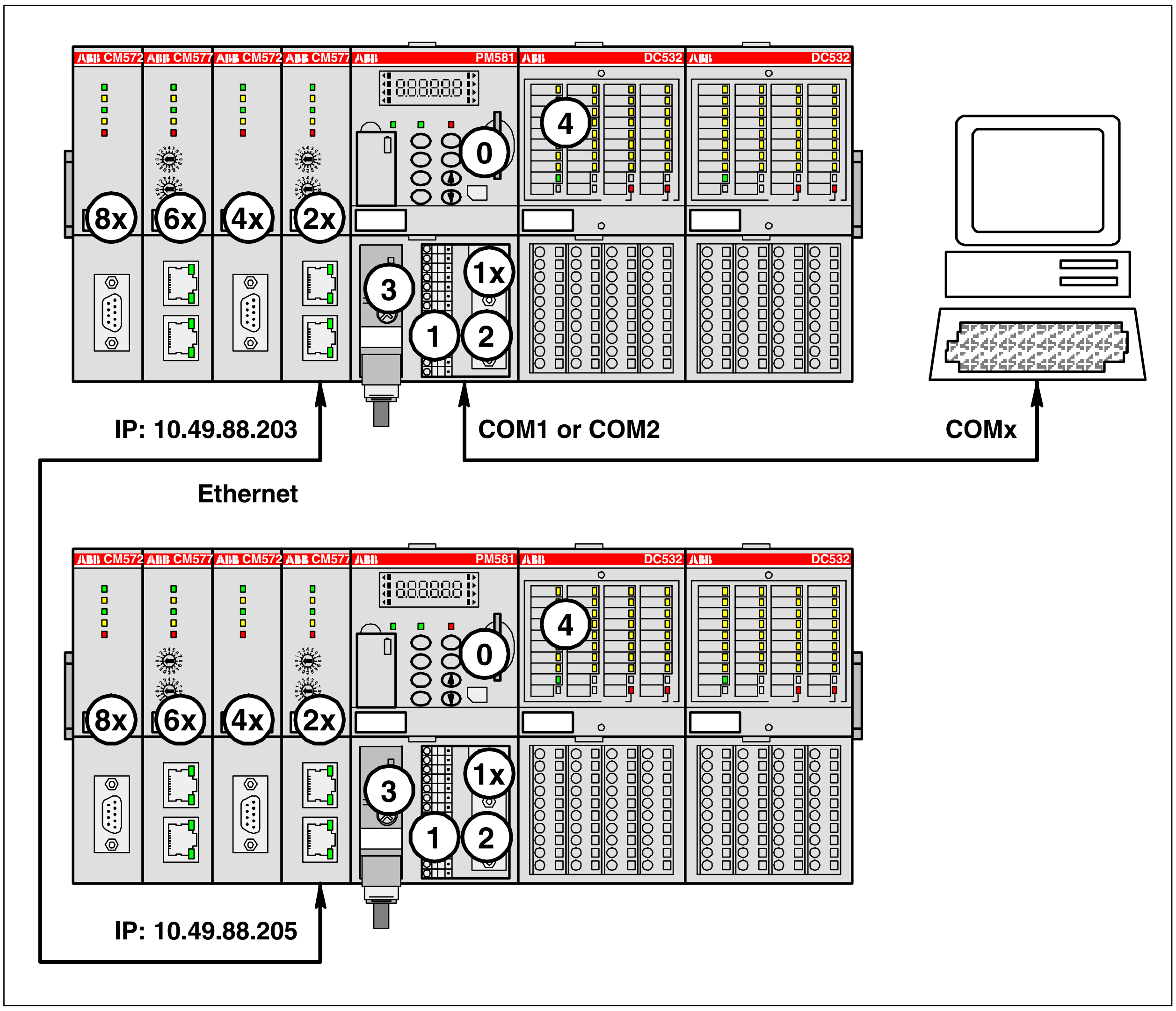
If you want to use the serial routing driver in order to directly access the connected CPU, set all routing parameters (parameter Routing levels and following parameters listed in the table above) to 0.

Note: Routing is available with version V1.3.x of the Control Builder and PLC firmware.

The following applies to the routing levels:

Routing levels = 0 No routing (parameters for Level 1 and Level 2 not set)
Routing levels = 1 Single-level routing (set parameters for Level 1)
Routing levels = 2 Double-level routing (set parameters for Level 1 and Level 2)

Example:



Configuration of PLC 2 (IP: 10.49.88.205) shall be performed via the external Ethernet coupler (IP: 10.49.88.203) inserted in slot 1 of PLC 1. The serial PC interface COM2 is connected to the serial interface COM1 of PLC 1:

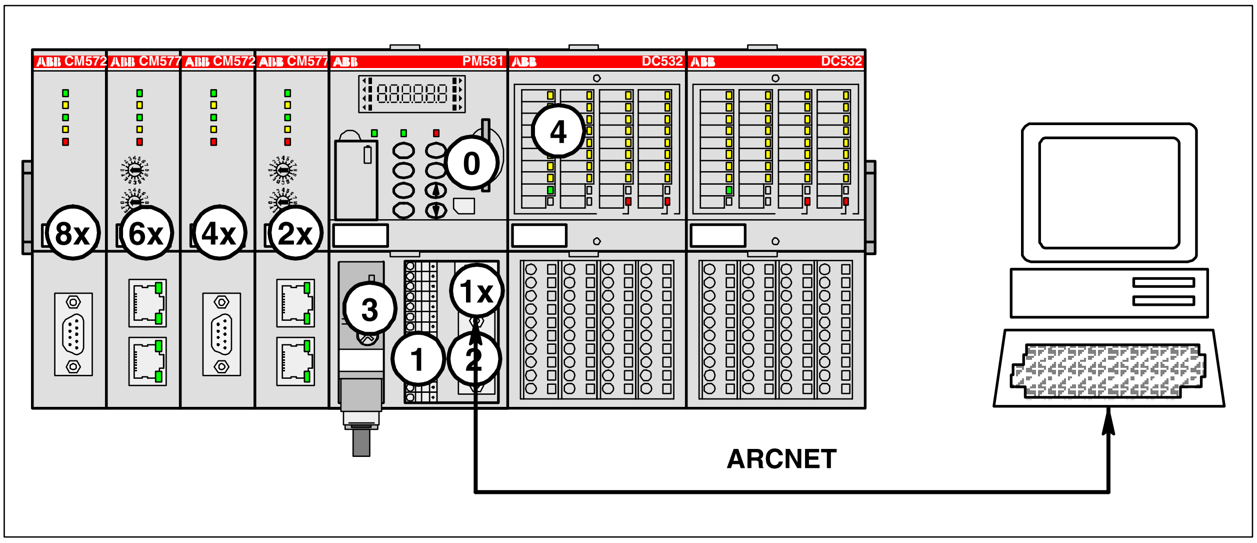
Parameter	Value	Remark
Port	COM2	PC COM2
Baudrate	19200	
Parity	No	
Stop bits	1	
Routing levels	1	Single-level routing
Coupler (Level 1)	Line 1	Coupler in slot 1
Channel (Level 1)	0	Channel 1
Address (Level 1)	10, 49, 88, 205, 0	Subscriber address of the target PLC (Node 2)
Coupler (Level 2)	0	No level 2
Channel (Level 2)	0	
Address (Level 2)	0, 0, 0, 0, 0	
Motorola byteorder	yes	Motorola byteorder

10.3 Programming via ARCNET

Note: The ARCNET coupler is available as of Control Builder version V1.2 and PLC firmware version V1.2.0.

Programming via ARCNET is only possible on a PC with installed ARCNET board. The installation of the board(s) and drivers is described in the Installation chapter (see also Installation / ARCNET drivers).

When programming via ARCNET, the PC is an ARCNET node.



The "sender node", i.e., the ARCNET subscriber address of the PC is set in the file:

Arcnet_xx.ini with the parameter NodeID1 = 254.

The file Arcnet_xx.ini is located in the folder where the PC operating system is installed (for example C:\WINNT for Win2000).

For a PC with installed ARCNET board, the file Arcnet_xx.ini contains for example the following entries:

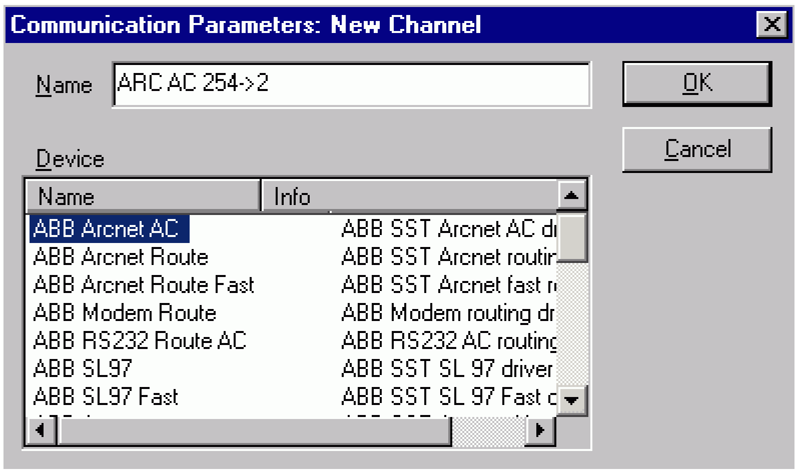
```
[ARCNET]
DriverAccessName1=FARC
;Default = Farc
NodeID1 = 254
; Default = 254

;DriverAccessName2=FARC1
; Default = Farc1
;NodeID2 = 253
; Default = 253

;DriverAccessName3=FARC11
; Default = Farc11
;NodeID3 = 252
; Default = 252

;DriverAccessName4=FARC111
; Default = Farc111
;NodeID4 = 251
; Default = 251
```

10.3.1 ARCNET driver "ABB Arcnet AC"

As of PLC runtime system version V1.2.0 and Control Builder version V1.2, the driver "ABB Arcnet AC" is available. This driver provides the following functions:

- Online operation of the PLC with the Control Builder
- Online operation of the PLC with the fieldbus configurator SYCON.net
- OPC connection with OPC server, as of version V1.3
- Parallel operation of Control Builder and SYCON.net
- Parallel operation of Control Builder and OPC server
- Parallel operation of Control Builder instances with several PLCs
- Online operation of AC31 series 90 controllers (07KT9x)

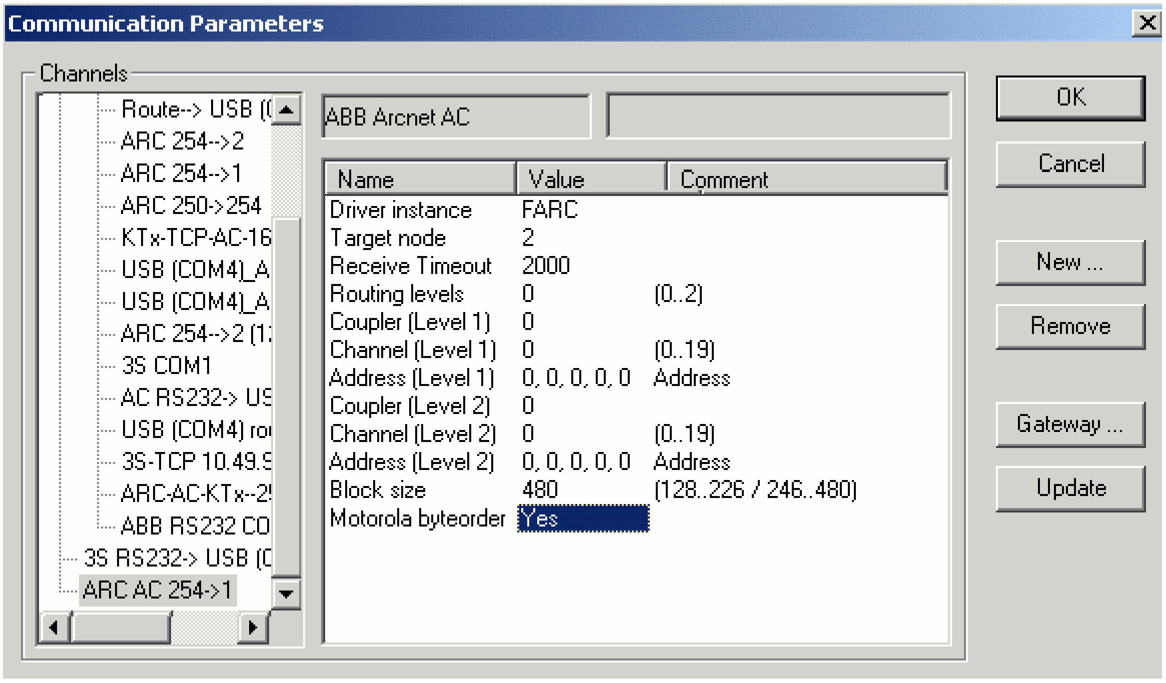
To define a new gateway channel for the ARCNET routing driver, select "Online/Communication Parameters" and press the button "New" in the "Communication Parameters" window. In the appearing window, enter a name for the channel (for example ARC AC 254 -> 2) and select the driver "ABB Arcnet AC" from the device list.



Name: ARC AC 254->2

Name	Info
ABB Arcnet AC	ABB SST Arcnet AC dı
ABB Arcnet Route	ABB SST Arcnet routir
ABB Arcnet Route Fast	ABB SST Arcnet fast rı
ABB Modem Route	ABB Modem routing dr
ABB RS232 Route AC	ABB RS232 AC routing
ABB SL97	ABB SST SL 97 driver
ABB SL97 Fast	ABB SST SL 97 Fast c

The following communication parameters can be set for the ARCNET routing driver "ABB Arcnet AC":



Name	Value	Comment
Driver instance	FARC	
Target node	2	
Receive Timeout	2000	
Routing levels	0	(0..2)
Coupler (Level 1)	0	
Channel (Level 1)	0	(0..19)
Address (Level 1)	0, 0, 0, 0, 0	Address
Coupler (Level 2)	0	
Channel (Level 2)	0	(0..19)
Address (Level 2)	0, 0, 0, 0, 0	Address
Block size	480	(128..226 / 246..480)
Motorola byteorder	Yes	

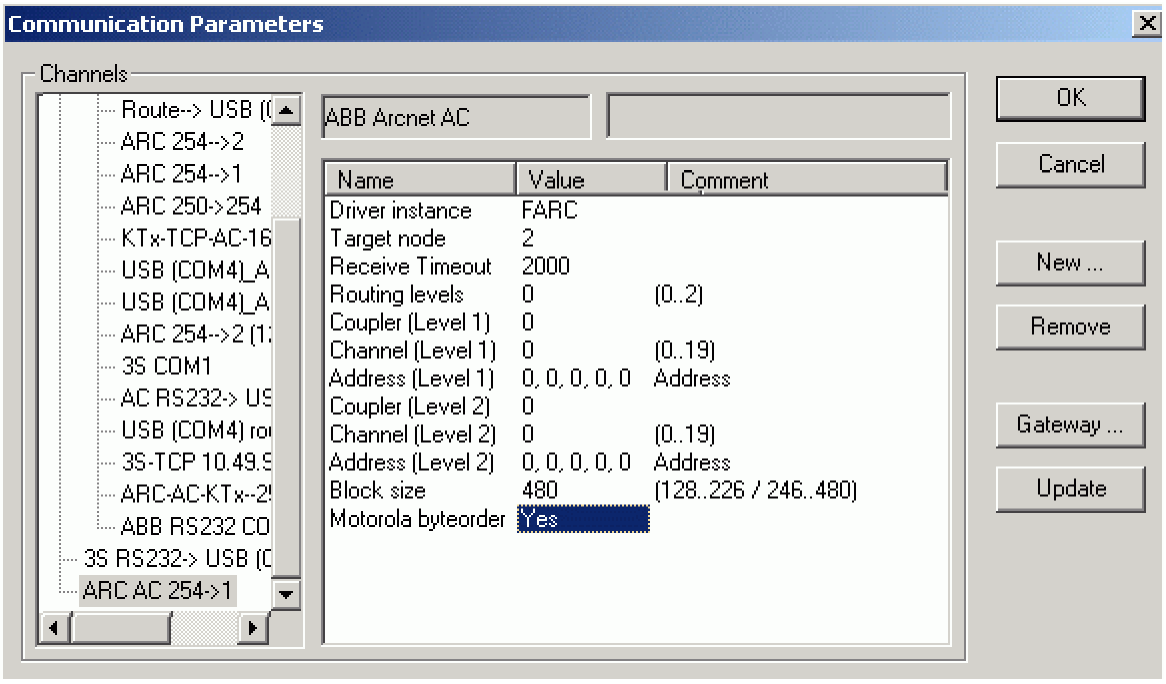
Parameter	Possible values	Meaning
Driver instance	FARC	DriverAccessName set in Arcnet_xx.ini
Target node	1...255	ARCNET subscriber address of the PLC
Receive Timeout	>= 2000	Timeout [ms] for response
Routing levels	0...2	Routing levels (0 = none)
Coupler (Level 1)	0, line 0...line 4	Coupler for level 1
Channel (Level 1)	0...19	Channel on coupler level 1
Address (Level 1)	0, 0, 0, 0, 0 (max. 5 bytes)	Address in target coupler level 1
Coupler (Level 2)	0, line 0...line 4	Coupler for level 2
Channel (Level 2)	0...19	Channel on coupler level 2
Address (Level 2)	0, 0, 0, 0, 0 (max. 5 bytes)	Address in target coupler level 2
Block size	128...226 / 246...480	User data size
Motorola byteorder	yes/no	Motorola or Intel byteorder

If you want to use the ARCNET routing driver to directly access the connected CPU, set all routing parameters (parameter Routing levels and following parameters listed in the table above) to 0.

The parameter "Block size" (128...480) sets the number of user data within one block. The default value is 480 (this is the maximum allowed block size). **Values in the range of 227 .. 245 are not allowed.**

Note: If you want to access a fieldbus coupler using SYCON.net via ARCNET, the parameter "Block size" always must be set to 128!

The parameter **"Motorola byteorder"** must be set to **"Yes"** for AC500 controllers.

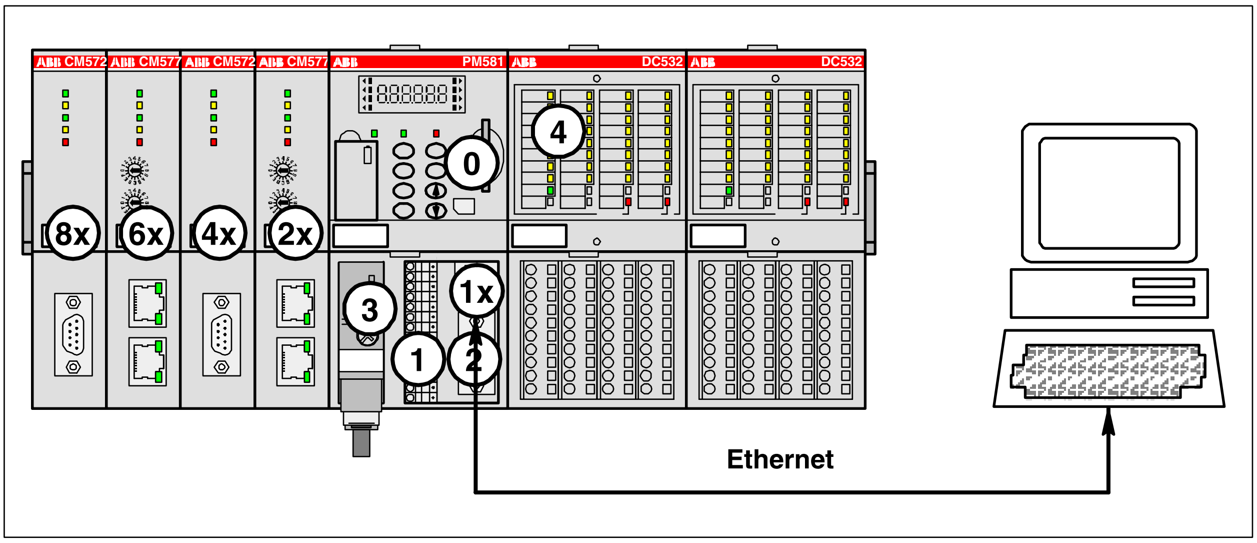


Name	Value	Comment
Driver instance	FARC	
Target node	2	
Receive Timeout	2000	
Routing levels	0	(0..2)
Coupler (Level 1)	0	
Channel (Level 1)	0	(0..19)
Address (Level 1)	0, 0, 0, 0, 0	Address
Coupler (Level 2)	0	
Channel (Level 2)	0	(0..19)
Address (Level 2)	0, 0, 0, 0, 0	Address
Block size	480	(128..226 / 246..480)
Motorola byteorder	Yes	

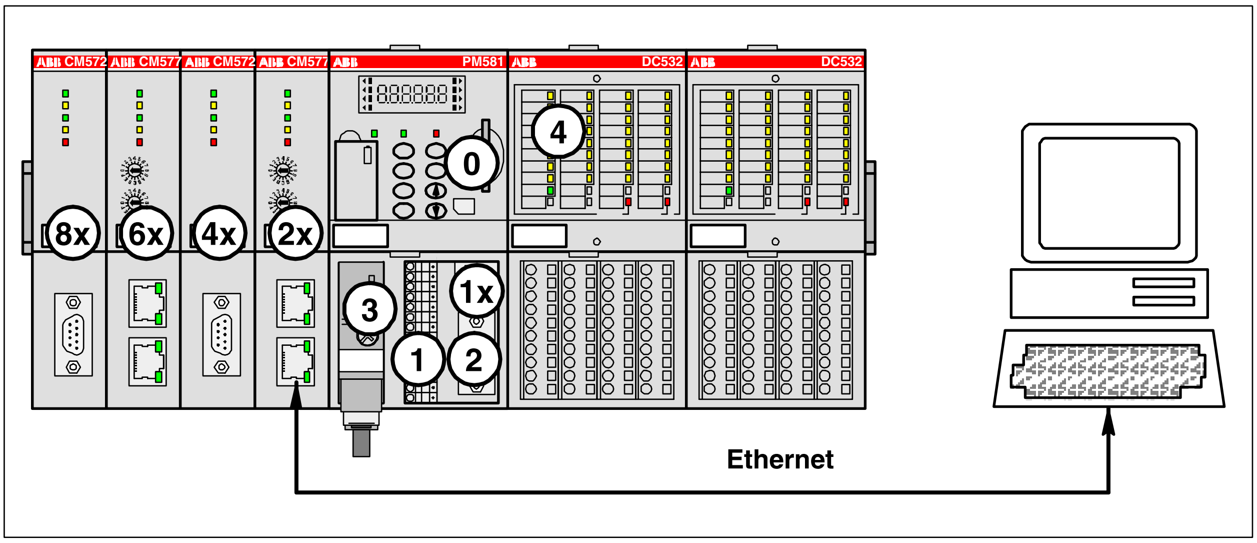
10.4 Programming via Ethernet (TCP/IP)

Programming via Ethernet is only possible on a PC with installed Ethernet board and installed network. Programming can be done via the internal and external Ethernet coupler.

Programming via internal Ethernet coupler:



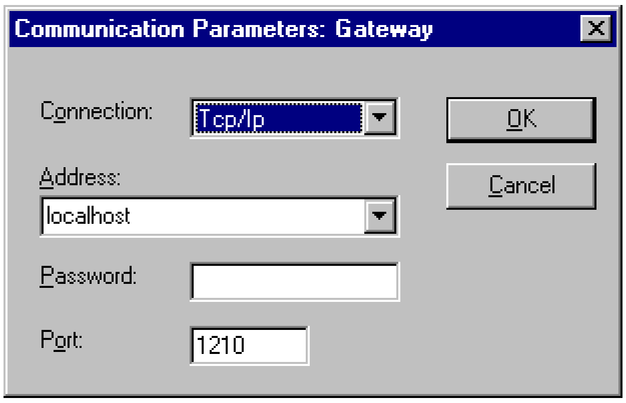
Programming via external Ethernet coupler (in this example coupler 1 in slot 1):



Note: Information how to set the IP addresses is available in the section "System Technology"=>"System technology of internal couplers"=>"The Ethernet coupler".

Gateway configuration for Ethernet:

In the Control Builder, select "Online/Communication Parameters" and press the button "Gateway" in the "Communication Parameters" window. In the appearing window, select "Tcp/Ip" from the "Connection" list box (see CoDeSys / chapter The Individual Components / Online Communication Parameters).


Communication Parameters: Gateway

Connection: Tcp/Ip

Address: localhost

Password:

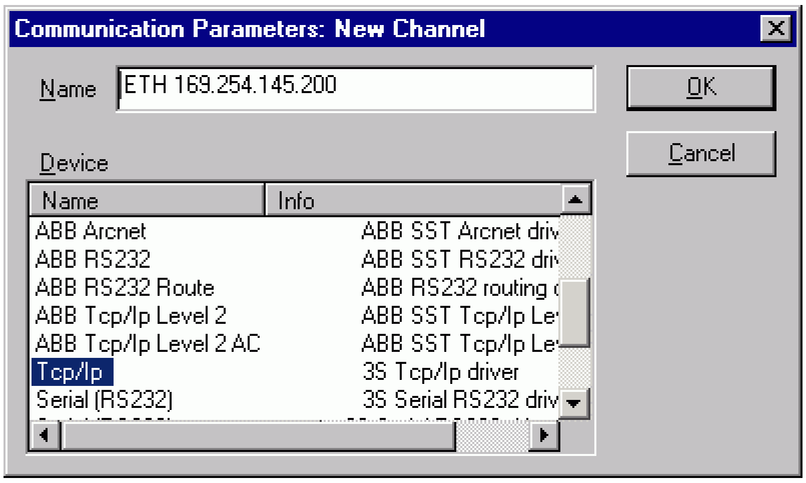
Port: 1210

10.4.1 Ethernet driver "Tcp/Ip"

Programming AC500 controllers with internal and/or external Ethernet coupler via Ethernet can be done by using the driver "Tcp/Ip". This driver provides the following functions:

- Online operation of the PLC with the Control Builder
- Online operation of the PLC with the fieldbus configurator SYCON.net
- OPC connection with OPC server, as of version V1.3
- Parallel operation of Control Builder and SYCON.net
- Parallel operation of Control Builder and OPC server
- Parallel operation of Control Builder instances with several PLCs

To define a new gateway channel for the Ethernet interface, select "Online/Communication Parameters" and press the button "New" in the "Communication Parameters" window. In the appearing window, enter a name for the channel (for example ETH 169.254.145.200) and select the driver "Tcp/Ip" from the device list.

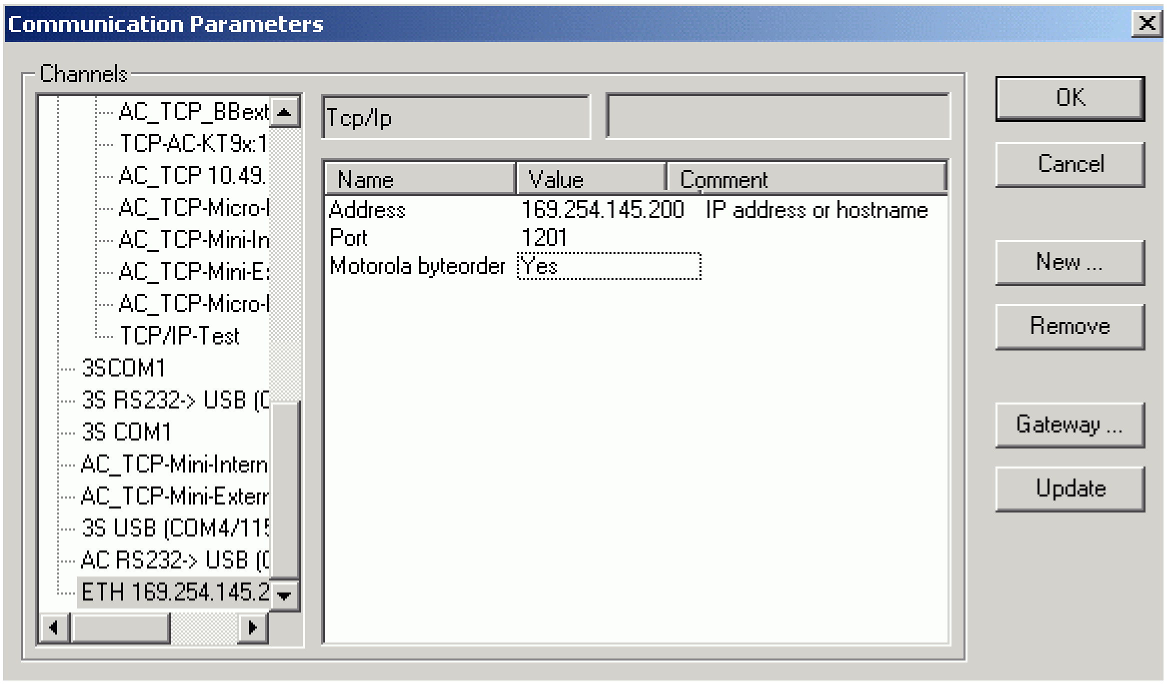

Communication Parameters: New Channel

Name: ETH 169.254.145.200

Device

Name	Info
ABB Arcnet	ABB SST Arcnet driv
ABB RS232	ABB SST RS232 driv
ABB RS232 Route	ABB RS232 routing d
ABB Tcp/Ip Level 2	ABB SST Tcp/Ip Le
ABB Tcp/Ip Level 2 AC	ABB SST Tcp/Ip Le
Tcp/Ip	3S Tcp/Ip driver
Serial (RS232)	3S Serial RS232 driv

The following communication parameters can be set for the Ethernet driver "Tcp/Ip":



Name	Value	Comment
Address	169.254.145.200	IP address or hostname
Port	1201	
Motorola byteorder	Yes	

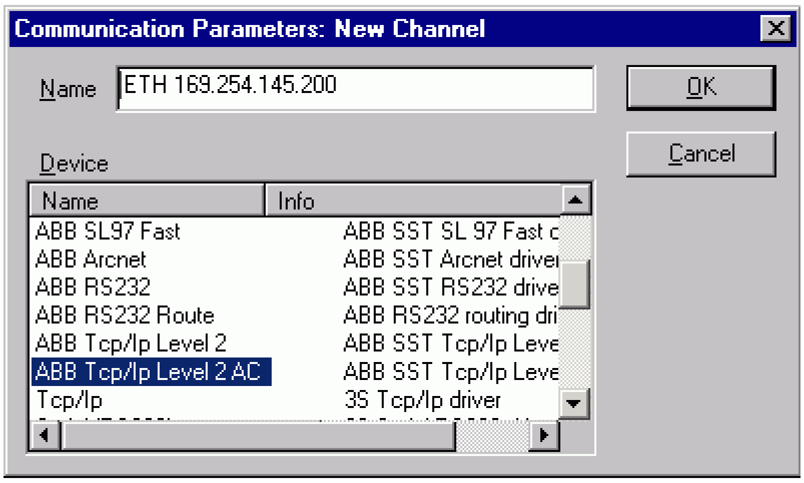
Parameter	Possible values	Meaning
Address	0.0.0.0	IP address or hostname of the PLC
Port	1201	Port 1201
Motorola byteorder	Yes (Yes/No)	Motorola or Intel byteorder (=Yes for AC500)

10.4.2 Ethernet driver "ABB Tcp/Ip Level 2 AC"

As of version V1.2, the driver "ABB Tcp/Ip Level 2 AC" is available for programming AC500 controllers with internal and/or external Ethernet coupler via Ethernet. This driver provides the following functions:

- Online operation of the PLC with the Control Builder
- Online operation of the PLC with the fieldbus configurator SYCON.net
- OPC connection with OPC server, as of version V1.3
- Parallel operation of Control Builder and SYCON.net
- Parallel operation of Control Builder and OPC server
- Parallel operation of Control Builder instances with several PLCs
- Online operation of PLCs connected via ARCNET. One PLC equipped with Ethernet coupler and one PLC with ARCNET coupler (Routing Ethernet -> ARCNET), as of version V2.x
- Online operation of AC31 series 90 controllers (07KT9x)

To define a new gateway channel for the Ethernet interface, select "Online/Communication Parameters" and press the button "New" in the "Communication Parameters" window. In the appearing window, enter a name for the channel (for example ETH 169.254.145.200) and select the driver "ABB Tcp/Ip Level 2 AC" from the device list.



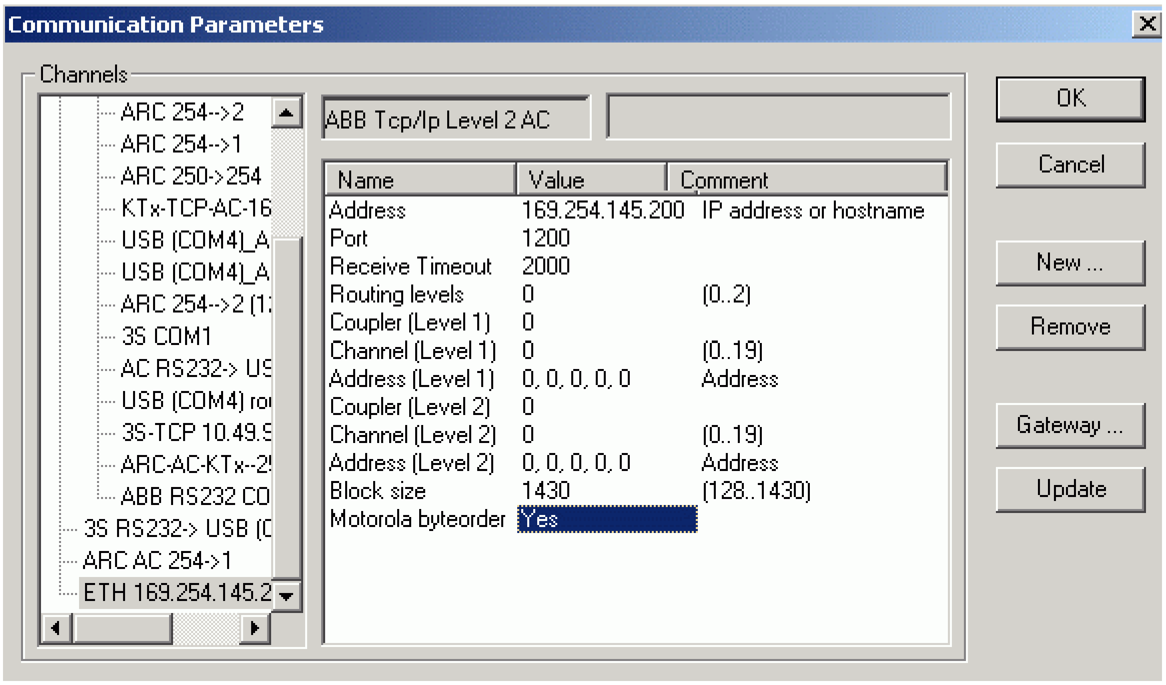
Communication Parameters: New Channel

Name: ETH 169.254.145.200

Device

Name	Info
ABB SL97 Fast	ABB SST SL 97 Fast c
ABB Arcnet	ABB SST Arcnet drive
ABB RS232	ABB SST RS232 drive
ABB RS232 Route	ABB RS232 routing dri
ABB Tcp/Ip Level 2	ABB SST Tcp/Ip Leve
ABB Tcp/Ip Level 2 AC	ABB SST Tcp/Ip Leve
Tcp/Ip	3S Tcp/Ip driver

The following communication parameters can be set for the Ethernet driver "ABB Tcp/Ip Level 2 AC":



Communication Parameters

Channels
- ARC 254-->2
- ARC 254-->1
- ARC 250->254
- KTx-TCP-AC-16
- USB (COM4)_A
- USB (COM4)_A
- ARC 254-->2 (1
- 3S COM1
- AC RS232-> US
- USB (COM4) ro
- 3S-TCP 10.49.9
- ARC-AC-KTx--2
- ABB RS232 CO
- 3S RS232-> USB (C
- ARC AC 254->1
- ETH 169.254.145.2

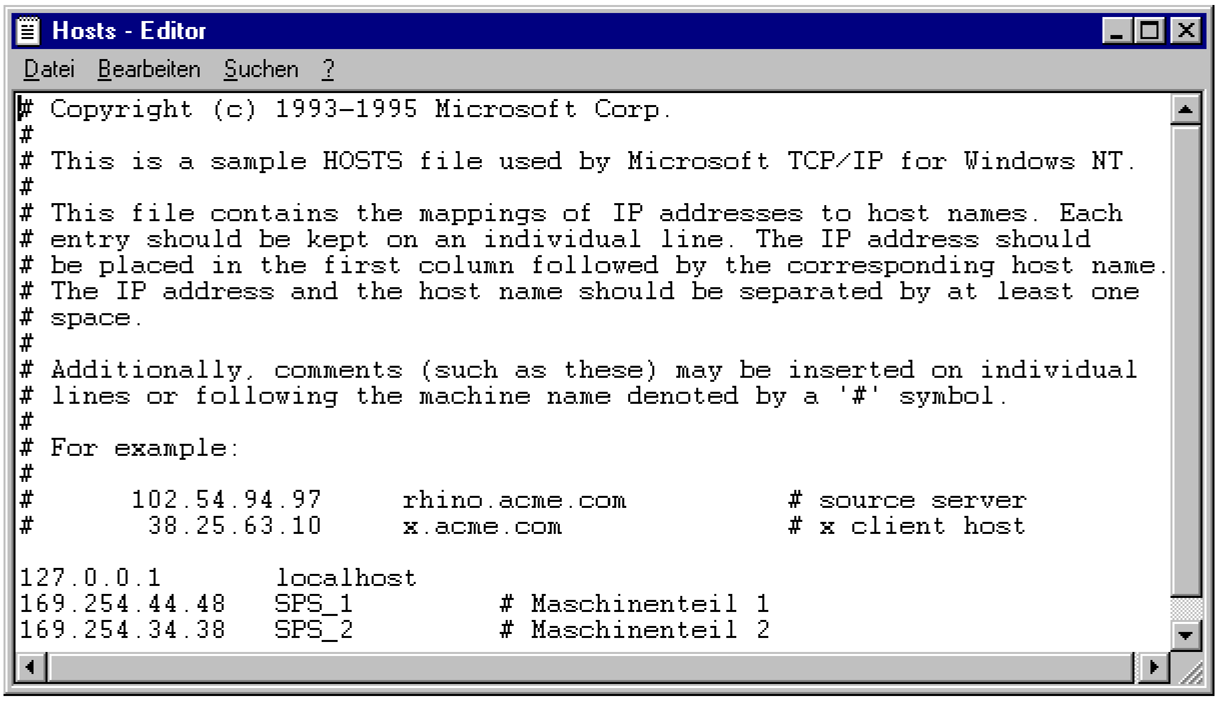
ABB Tcp/Ip Level 2 AC

Name	Value	Comment
Address	169.254.145.200	IP address or hostname
Port	1200	
Receive Timeout	2000	
Routing levels	0	(0..2)
Coupler (Level 1)	0	
Channel (Level 1)	0	(0..19)
Address (Level 1)	0, 0, 0, 0, 0	Address
Coupler (Level 2)	0	
Channel (Level 2)	0	(0..19)
Address (Level 2)	0, 0, 0, 0, 0	Address
Block size	1430	(128..1430)
Motorola byteorder	Yes	

Parameter	Possible values	Meaning
Address	0.0.0.0	IP address or hostname of the PLC
Port	1200	Port 1200
Timeout (ms)	>= 2000	Timeout [ms] for response
Routing levels	0...2	Routing levels (0 = none)
Coupler (Level 1)	0, line 0...line 4	Coupler for level 1
Channel (Level 1)	0...19	Channel on coupler level 1
Address (Level 1)	0, 0, 0, 0, 0 (max. 5 bytes)	Address in target coupler level 1
Coupler (Level 2)	0, line 0...line 4	Coupler for level 2
Channel (Level 2)	0...19	Channel on coupler level 2
Address (Level 2)	0, 0, 0, 0, 0 (max. 5 bytes)	Address in target coupler level 2
Block size	1430 (128...1430)	Bytes per telegram (unallowed 227..245)
Motorola byteorder	Yes (Yes/No)	Motorola or Intel byteorder (=Yes for AC500)

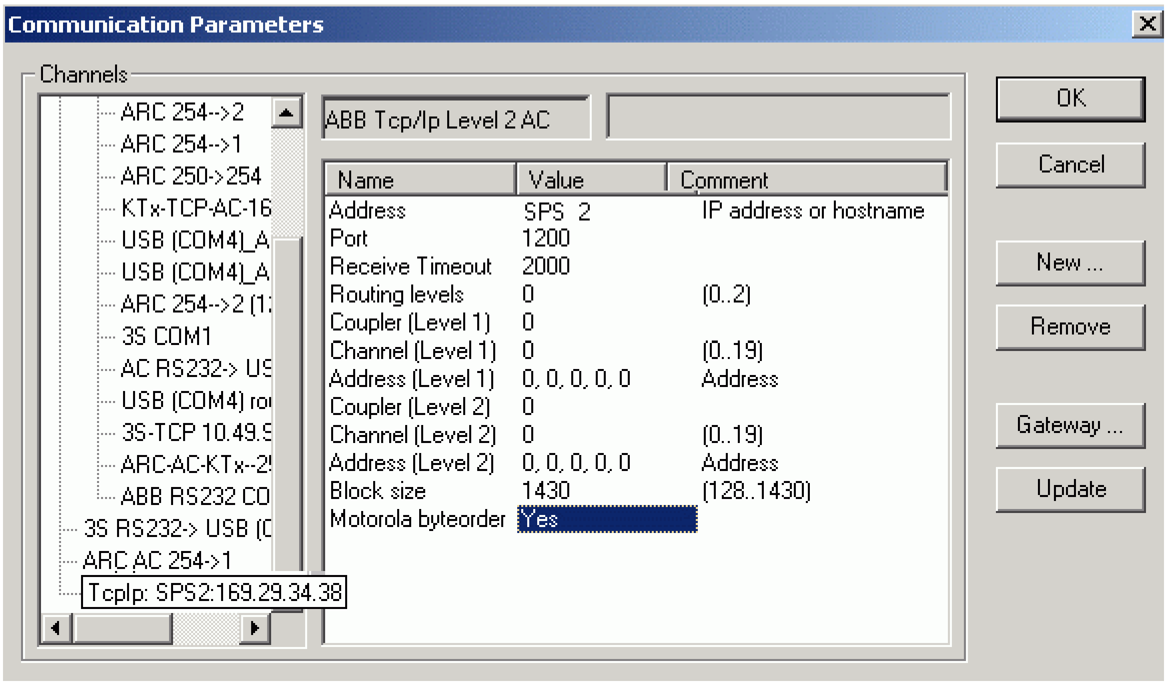
If you want to use the Ethernet driver to directly access the PLC, set all routing parameters (parameter Routing levels and following parameters listed in the table above) to 0.

The "Address" parameter sets the IP address or hostname of the PLC. To be able to use hostnames, the names have to be added to the file "Hosts". Under Win2000, this file is located in the directory "WINNT\System32\drivers\etc".



```
# Copyright (c) 1993-1995 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows NT.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#      102.54.94.97     rhino.acme.com          # source server
#       38.25.63.10     x.acme.com              # x client host

127.0.0.1       localhost
169.254.44.48   SPS_1          # Maschinenteil 1
169.254.34.38   SPS_2          # Maschinenteil 2
```

If you have changed the "Hosts" file accordingly, you can enter the symbolic name for the "Address" parameter instead of the IP address. In the following figure, the IP address "169.254.34.38" is replaced by the hostname "SPS_2".

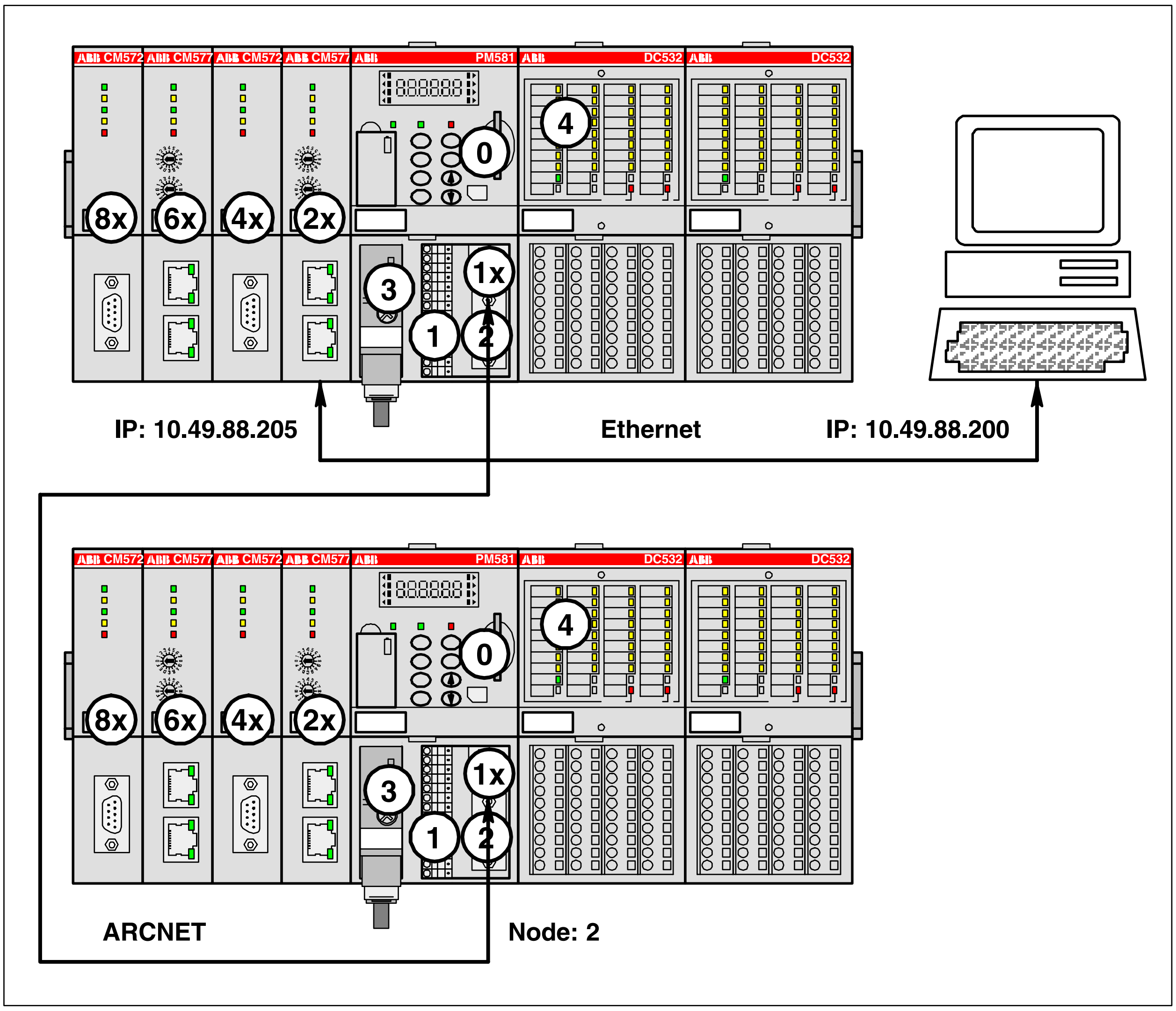


Name	Value	Comment
Address	SPS_2	IP address or hostname
Port	1200	
Receive Timeout	2000	
Routing levels	0	(0..2)
Coupler (Level 1)	0	
Channel (Level 1)	0	(0..19)
Address (Level 1)	0, 0, 0, 0, 0	Address
Coupler (Level 2)	0	
Channel (Level 2)	0	(0..19)
Address (Level 2)	0, 0, 0, 0, 0	Address
Block size	1430	(128..1430)
Motorola byteorder	Yes	

10.4.3 Ethernet ARCNET routing

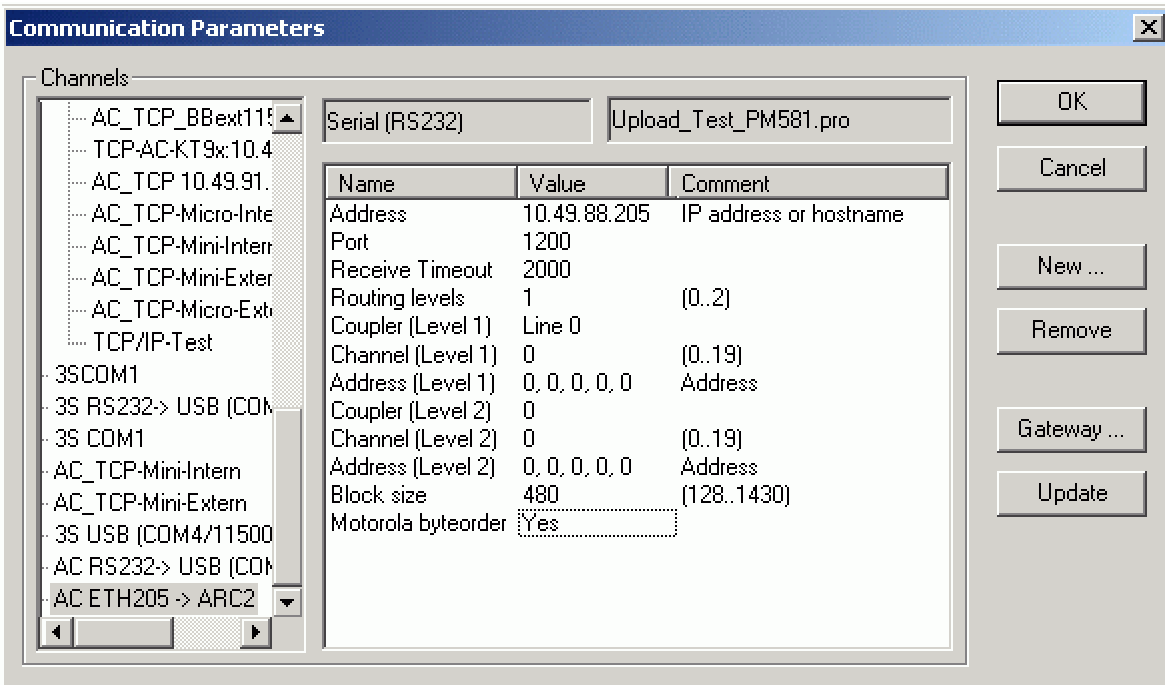
Note: Routing is available as of PLC firmware version V1.3.

For controllers with Ethernet and ARCNET coupler, the PLCs connected via ARCNET can be programmed using the PLC Ethernet interface.



For each PLC connected via ARCNET, one gateway channel has to be defined. To do this, select "Online/Communication Parameters" and press the button "New" in the "Communication Parameters" window. In the appearing window, enter a name for the channel (for example TcpIp: PLC1:169.29.44.48 -> ARC_2) and select the driver "ABB Tcp/Ip Level 2 AC" from the device list.

For example, set the communication parameters as follows for the configuration shown above:



Name	Value	Comment
Address	10.49.88.205	IP address or hostname
Port	1200	
Receive Timeout	2000	
Routing levels	1	(0..2)
Coupler (Level 1)	Line 0	
Channel (Level 1)	0	(0..19)
Address (Level 1)	0, 0, 0, 0, 0	Address
Coupler (Level 2)	0	
Channel (Level 2)	0	(0..19)
Address (Level 2)	0, 0, 0, 0, 0	Address
Block size	480	(128..1430)
Motorola byteorder	Yes	

Parameter	Possible values	Meaning
Address	10.49.88.205	IP address of PLC 1
Port	1200	Port 1200
Timeout (ms)	2000	Timeout [ms] for response
Routing levels	1	Single-level routing
Coupler (Level 1)	Line 0	Coupler for level 1 (internal: ARCNET)
Channel (Level 1)	0	Channel on coupler level 1
Address (Level 1)	2, 0, 0, 0, 0	ARCNET node of the target PLC (Node 2)
Coupler (Level 2)	0	No level 2
Channel (Level 2)	0	
Address (Level 2)	0, 0, 0, 0, 0	
Block size	480	Bytes per block: 128...1430
Motorola byteorder	Yes	

For the parameter "Coupler (Level 1)", enter the slot where the ARCNET coupler "Line 0" is inserted (the ARCNET coupler is always the internal coupler).

The ARCNET coupler has only one communication channel. Thus, the "Channel" value must always be 0.

For the ARCNET coupler, 1 byte is required for the subscriber address (node). The address (Node=2) of the target PLC is entered to the first byte of the address byte.

The default value for the block size is 1430. If routing on ARCNET is required (and "large ARCNET packages" are enabled for the target PLC), the block size can be increased to 480 bytes. **Values in the range of 227 .. 245 are not allowed.**

11 Communication with Modbus RTU

11.1 Protocol description

The Modbus protocol is used worldwide. The **MODICON Modbus® RTU** protocol is implemented in the AC500 CPU.

Numerous automation devices, such as PLC installations, displays, variable-frequency inverters or monitoring systems have a Modbus® RTU interface by default or as an option and can therefore communicate with AC500 basic units without any problems.

Modbus® is a master-slave protocol. The master sends a request to the slave and receives its response.

Modbus master

In operating mode MODBUS master, the telegram traffic with the slave(s) is handled via the function block MODMAST. The function block MODMAST sends Modbus request telegrams to the slave via the set interface and receives Modbus response telegrams from the slave via this interface.

For Modbus on TCP/IP, the function block ETH_MODMAST is used and for serial interfaces the function block COM_MODMAST (link to function blocks: ETH_MODMAST in library Ethernet_AC500_Vxx.lib and COM_MODMAST in library Modbus_AC500_Vxx.lib).

The Modbus® blocks transferred by the master contain the following information:

- Modbus® address of the interrogated slave (1 byte)
- Function code that defines the request of the master (1 byte)
- Data to be exchanged (n bytes)
- CRC16 control code (2 bytes)

Modbus slave

In operating mode MODBUS slave, no function block is required for Modbus communication. Sending and receiving Modbus telegrams is performed automatically.

The AC500 CPUs process only the following Modbus® operation codes:

Function code		**Description**
DEC	**HEX**	
01 or 02	01 or 02	read n bits
03 or 04	03 or 04	read n words
05	05	write one bit
06	06	write one word
07	07	fast reading the status byte of the CPU
15	0F	write n bits
16	10	write n words

The following restrictions apply to the length of the data to be sent:

Function code		Max. length	
DEC	**HEX**	**Serial**	**Modbus on TCP/IP**
01 or 02	01 or 02	2000 bits	255 bits (up to coupler FW V01.033) xxx bits (as of coupler FW V01.041)
03 or 04	03 or 04	125 words / 62 double words	100 words / 50 double words
05	05	1 bit	1 bit
06	06	1 word	1 word
07	07	8 bits	8 bits
15	0F	1968 bits	255 bits (up to coupler FW V01.033) xxx bits (as of coupler FW V01.041)
16	10	123 words / 61 double words	100 words / 50 double words

11.2 Modbus RTU with the serial interfaces COM1 and COM2

11.2.1 Modbus operating modes of the serial interfaces

Both serial interfaces of the AC500 CPUs can be operated simultaneously as Modbus interfaces and can operate as Modbus master as well as Modbus slave.

The Modbus operating mode and the interface parameters are set in the PLC Configuration (see also Controller configuration / Modbus).

Description of the Modbus® protocol:

Supported standard	EIA RS-232 / RS-485
Number of connection points	1 master max. 1 slave with RS 232 interface max. 31 slaves with RS 485
Protocol	Modbus® (Master/Slave)
Data transmission control	CRC16
Data transmission speed	up to 187500 baud
Encoding	1 start bit 8 data bits 1 parity bit, even or odd (optional) 1 or 2 stop bits
Max. cable length	for RS 485: 1200 m at 19200 baud

11.3 Modbus on TCP/IP via Ethernet

Modbus on TCP/IP is described in the chapter System Technology Coupler / The Ethernet coupler (see also System Technology Ethernet Coupler / Modbus on TCP/IP).

11.4 Modbus addresses

11.4.1 Modbus address table

A range of 128 kbytes is allowed for the access via Modbus, i.e., the segments line 0 and line 1 of the addressable flag area (%M area) can be accessed. Thus, the complete address range 0000hex up to FFFFhex is available for Modbus.

The availability of the segments depends on the CPU. The size of the %M area can be found in the technical data of the CPUs (see Technical data of the CPUs) and in the target system settings (see Target Support Package).

Inputs and outputs cannot be directly accessed using Modbus.

The address assignment for word and double word accesses is done according to the following table:

Modbus address		**Byte** **BYTE**	**Bit (byte-oriented)** **BOOL**	**Word** **WORD**	**Double word** **DWORD**
HEX	**DEC**				
Line 0					
0000	0	%MB0.0	%MX0.0.0...%MX0.0.7	**%MW0.0**	**%MD0.0**
		%MB0.1	%MX0.1.0...%MX0.1.7		
0001	1	%MB0.2	%MX0.2.0...%MX0.2.7	**%MW0.1**	
		%MB0.3	%MX0.3.0...%MX0.3.7		
0002	2	%MB0.4	%MX0.4.0...%MX0.4.7	**%MW0.2**	**%MD0.1**
		%MB0.5	%MX0.5.0...%MX0.5.7		
0003	3	%MB0.6	%MX0.6.0...%MX0.6.7	**%MW0.3**	
		%MB0.7	%MX0.7.0...%MX0.7.7		
...					
7FFE	32766	%MB0.65532	%MX0.65532.0 ...%MX0.65532.7	**%MW0.32766**	**%MD0.16383**
		%MB0.65533	%MX0.65533.0 ...%MX0.65533.7		
7FFF	32767	%MB0.65534	%MX0.65534.0 ...%MX0.65534.7	**%MW0.32767**	
		%MB0.65535	%MX0.65535.0 ...%MX0.65535.7		
Line 1					
8000	32768	%MB1.0	%MX1.0.0...%MX1.0.7	**%MW1.0**	**%MD1.0**
		%MB1.1	%MX1.1.0...%MX1.1.7		
8001	32769	%MB1.2	%MX1.2.0...%MX1.2.7	**%MW1.1**	
		%MB1.3	%MX1.3.0...%MX1.3.7		
8002	32770	%MB1.4	%MX1.4.0...%MX1.4.7	**%MW1.2**	**%MD1.1**
		%MB1.5	%MX1.5.0...%MX1.5.7		
8003	32771	%MB1.6	%MX1.6.0...%MX1.6.7	**%MW1.3**	
		%MB1.7	%MX1.7.0...%MX1.7.7		
...					
FFFE	65534	%MB1.65532	%MX1.65532.0 ...%MX1.65532.7	**%MW1.32766**	**%MD1.16383**
		%MB1.65533	%MX1.65533.0 ...%MX1.65533.7		
FFFF	65535	%MB1.65534	%MX1.65534.0 ...%MX1.65534.7	**%MW1.32767**	
		%MB1.65535	%MX1.65535.0 ...%MX1.65535.7		

The **address assignment for bit accesses** is done according to the following table:

Modbus address		**Byte BYTE**	**Bit (byte-oriented) BOOL**	**Word WORD**	**Double word DWORD**
HEX	**DEC**				
Line 0					
0000	0	%MB0.0	**%MX0.0.0**	%MW0.0	%MD0.0
0001	1		**%MX0.0.1**		
0002	2		**%MX0.0.2**		
0003	3		**%MX0.0.3**		
0004	4		**%MX0.0.4**		
0005	5		**%MX0.0.5**		
0006	6		**%MX0.0.6**		
0007	7		**%MX0.0.7**		
0008	8	%MB0.1	**%MX0.1.0**		
0009	9		**%MX0.1.1**		
000A	10		**%MX0.1.2**		
000B	11		**%MX0.1.3**		
000C	12		**%MX0.1.4**		
000D	13		**%MX0.1.5**		
000E	14		**%MX0.1.6**		
000F	15		**%MX0.1.7**		
0010	16	%MB0.2	**%MX0.2.0**	%MW0.1	
0011	17		**%MX0.2.1**		
0012	18		**%MX0.2.2**		
0013	19		**%MX0.2.3**		
0014	20		**%MX0.2.4**		
0015	21		**%MX0.2.5**		
0016	22		**%MX0.2.6**		
0017	23		**%MX0.2.7**		
0018	24	%MB0.3	**%MX0.3.0**		
0019	25		**%MX0.3.1**		
001A	26		**%MX0.3.2**		
001B	27		**%MX0.3.3**		
001C	28		**%MX0.3.4**		
001D	29		**%MX0.3.5**		
001E	30		**%MX0.3.6**		
001F	31		**%MX0.3.7**		
0020	32	%MB0.4	**%MX0.4.0**	%MW0.2	%MD0.1
0021	33		**%MX0.4.1**		
0022	34		**%MX0.4.2**		
...	...	...	**...**	...	...
0FFF	4095	%MB0.511	**%MX0.511.7**	%MW0.255	%MD0.127
1000	4096	%MB0.512	**%MX0.512.0**	%MW0.256	%MD0.128
...	...	...	**...**	...	...
7FFF	32767	%MB0.4095	**%MX0.4095.7**	%MW0.2047	%MD0.1023
8000	32768	%MB0.4096	**%MX0.4096.0**	%MW0.2048	%MD0.1024
...	...	...	**...**	...	...
FFFF	65535	%MB0.8191	**%MX0.8191.7**	%MW0.4095	%MD0.2047

Calculation of the bit variable from the hexadecimal address:

Formula:			
	Bit variable (BOOL) := %MX0.BYTE.BIT		
where:	DEC	Decimal address	
	BYTE	DEC / 8	
	BIT	DEC mod 8	(Modulo division)

Examples:

Address hexadecimal = 16#2002
DEC := HEX2DEC(16#2002) := 8194
BYTE := 8194 / 8 := 1024
BIT := 8194 mod 8 := 2
Bit variable: %MX0.1024.2

Address hexadecimal = 16#3016
DEC := HEX2DEC(16#3016) := 12310
BYTE := 12310 / 8 := 1538,75 -> 1538
BIT := 12310 mod 8 := 6
Bit variable: %MX0.1538.6

Address hexadecimal = 16#55AA
DEC := HEX2DEC(16#55AA) := 21930
BYTE := 21930 / 8 := 2741,25 -> 2741
BIT := 21930 mod 8 := 2
Bit variable: %MX0.2741.2

Calculation of the hexadecimal address from the bit variable:

Formula:

Address hexadecimal := DEC2HEX(BYTE * 8 + BIT)

Examples:

Bit variable := %MX0.515.4
Address hex := DEC2HEX(515 * 8 + 4) := DEC2HEX(4124) := 16#101C

Bit variable := %MX0.3.3
Address hex := DEC2HEX(3 * 8 + 3) := DEC2HEX(27) := 16#001B

Bit variable := %MX0.6666.2
Address hex := DEC2HEX(6666 * 8 + 2) := DEC2HEX(53330) := 16#D052

11.4.2 Peculiarities for accessing Modbus addresses

Peculiarities for bit access:

- As you can see in the address table, a WORD in the %M area is assigned to each Modbus address 0000hex .. FFFFhex
- Bit addresses 0000hex .. FFFFhex are contained in the word range %MW0.0 .. %MW0.4095

Write/read-protected areas for the Modbus slave:

As described in the PLC configuration, one write-protected and one read-protected area can be defined for each segment line 0 and line 1. (see also Controller configuration / The setting 'COMx - Modbus'). If you try to write to a write-protected area or to read from a read-protected area, an error message is generated.

Segment exceedance for line 0 and line 1:

A write- or read-protected area that lies in both segments, line 0 and line 1, cannot be accessed with a write/read operation. In case of a segment exceedance, an error message is generated.

Example:
Read 10 words beginning at address := $7FFE_{hex}$
This includes the addresses: $7FFE_{hex}$...8007_{hex} with the operands %MW0.32766...%MW1.7. Because line 0 is exceeded in this case, an error message is generated.
Due to this, two telegrams have to be generated here:
1. Read 2 words beginning at address := $7FFE_{hex}$ and
2. Read 8 words beginning at address := 8000_{hex}.

Valid data areas for reading/writing the Modbus master:

If the AC500 control system operates as Modbus master, the data exchange with the Modbus slaves is controlled using a MODMAST block (ETH_MOD_MAST for Modbus on TCP/IP and COM_MOD_MAST for serial interfaces). (Link to blocks: ETH_MODMAST in library Ethernet_AC500_Vxx.lib and COM_MODMAST in library Modbus_AC500_Vxx.lib).

The address of the area from which data are to be read or to which data are to be written is specified at block input "Data" via the ADR operator.

For the AC500, the following areas can be accessed using the ADR operator:

- Inputs area (%I area)
- Outputs area (%Q area)
- Area of non-buffered variables (VAR .. END_VAR or VAR_GLOBAL END_VAR)
- Addressable flag area (also protected areas for %M area)
- Area of buffered variables (VAR RETAIN .. END_VAR or VAR_GLOBAL RETAIN .. END_VAR)

11.4.3 Comparison between AC500 and AC31/S90 Modbus addresses

The following table shows the addresses for AC500 controllers and its predecessor AC31 / S90

Address HEX	FCT HEX	AC1131 operand	FCT HEX	AC500 operand
Bit accesses				
0000...0FFF	01, 02	%IX0.0...%IX255.15	01, 02, 05, 07, 0F	%MX0.0.0...%MX0.511.7
0000		%IX0.0		%MX0.0.0
0001		%IX0.1		%MX0.0.1
0002		%IX0.2		%MX0.0.2
...		...		...
0010		%IX1.0		%MX0.2.0
...		...		...
0FFF		%IX255.15		%MX0.511.7
1000...1FFF	01, 02, 05, 0F	%QX0.0...%QX255.15	01, 02, 05, 07, 0F	%MX0.512.0...%MX0.1023.7
1000		%QX0.0		%MX0.512.0
1001		%QX0.1		%MX0.512.1
1002		%QX0.2		%MX0.512.2
...		...		...
1010		%QX1.0		%MX0.514.0
...		...		...
1FFF		%QX255.15		%MX0.1023.7
2000...2FFF	01, 02, 05, 07, 0F	%MX0.0...%MX255.15	01, 02, 05, 07, 0F	%MX0.1024.0...%MX0.1535.7
2000		%MX0.0		%MX0.1024.0
2001		%MX0.1		%MX0.1024.1
2002		%MX0.2		%MX0.1024.2
...		...		...
2010		%MX1.0		%MX0.1026.0
...		...		...
2FFF		%MX255.15		%MX0.1535.7
3000...3FFF	01, 02, 05, 07, 0F	%MX5000.0...%MX5255.15	01, 02, 05, 07, 0F	%MX0.1536.0...%MX0.2047.7
3000		%MX5000.0		%MX0.1536.0
3001		%MX5000.1		%MX0.1536.1
3002		%MX5000.2		%MX0.1536.2
...		...		...
3010		%MX5001.0		%MX0.1538.0
...		...		...
3FFF		%MX5255.15		%MX0.2047.7
4000...FFFF		No access	01, 02, 05, 07, 0F	%MX0.2048.0...%MX0.8191.7
Word accesses				
0000...0CFF	03, 04	%IW1000.0...%IW1207.15	03, 04, 06, 10	%MW0.0...%MW0.3327
0D00...0FFF	03, 04	No access	03, 04, 06, 10	%MW0.3328...%MW0.4095
1000...1CFF	03, 04, 06, 10	%QW1000.0...%QW1207.15	03, 04, 06, 10	%MW0.4096...%MW0.7423
1D00...1FFF		No access	03, 04, 06, 10	%MW0.7424...%MW0.8191
2000...2FFF	03, 04, 06, 10	%MW1000.0...%MW1255.15	03, 04, 06, 10	%MW0.8192...%MW0.12287
3000...359F	03, 04, 06, 10	%MW3000.0...%MW3089.15	03, 04, 06, 10	%MW0.12288...%MW0.13727
35A0...3FFF		No access	03, 04, 06, 10	%MW0.13728...%MW0.16383
4000...47FF		%MW2000.0.0...%MW2063.15.1 No access	03, 04, 06, 10	%MW0.16384...%MW18431
4800...4FFF		No access	03, 04, 06, 10	%MW0.18432...%MW0.20479
5000...517F		%MW4000.0.0...%MW4023.15.1 No access	03, 04, 06, 10	%MW0.20480...%MW0.21247
5180...FFFF		No access	03, 04, 06, 10	%MW0.21248...%MW1.32767
Double word accesses				
0000...3FFF		No access	03, 04, 06, 10	%MD0.0...%MD0.8191
4000...47FF	03, 04, 06, 10	%MD2000.0...%MD2063.15	03, 04, 06, 10	%MD0.8192...%MD0.9215
4800...4FFF		No access	03, 04, 06, 10	%MD0.9216...%MD0.10239
5000...537F	03, 04, 06, 10	%MD4000.0...%MD4023.15	03, 04, 06, 10	%MD0.1240...%MD0.10815
5480...FFFF		No access	03, 04, 06, 10	%MD0.10816...%MD1.16383

11.5 Modbus telegrams

The send and receive telegrams shown in this section are not visible in the PLC. However, the complete telegrams can be made visible using a serial data analyzer connected to the connection line between master and slave, if required.

The amount of user data depends on the properties of the master and slave.

For the following examples, it is assumed that an AC500 Modbus module is used as slave. There may be different properties if modules of other manufacturers are used.

FCT 1 or 2: Read n bits

Master request

Slave address	Function code	Slave operand address		Number of bits		CRC	
		High	Low	High	Low	High	Low

Slave response

Slave address	Function code	Number of bytes	...Data...	CRC	
				High	Low

Example:	Modbus interface of the master:	COM1
	Master reads from:	Slave 1
	Data:	%MX0.1026.4 = FALSE; %MX0.1026.5 = TRUE %MX0.1026.6 = FALSE
	Source address at slave:	%MX0.1026.4 : 2014HEX = 8212DEC
	Target address at master:	abReadBit : ARRAY[0..2] OF BOOL;
	The values of the flags %MX0.1026.4..%MX0.1026.6 on the slave are written to the ARRAY abReadBool on the master.	

Modbus request of the master

Slave address	Function code	Slave operand address		Number of bits		CRC	
		High	Low	High	Low	High	Low
01HEX	01HEX	20HEX	14HEX	00HEX	03HEX	37HEX	CFHEX

Modbus response of the slave

Slave address	Function code	Number of bytes	Data	CRC	
				High	Low
01HEX	01HEX	01HEX	02HEX	D0HEX	49HEX

Parameterization of the COM_MOD_MAST block inputs
NB = Number of bits

EN	COM	SLAVE	FCT	TIMEOUT	ADDR	NB	DATA
FALSE -> TRUE	1	1	1	Application-specific	8212	3	ADR (abReadBool[0])

FCT 3 or 4: Read n words

Master request

Slave address	Function code	Slave operand address		Number of words		CRC	
		High	Low	High	Low	High	Low

Slave response

Slave address	Function code	Number of bytes	...Data...	CRC	
				High	Low

Example:	Modbus interface of the master:	COM1
	Master reads from:	Slave 1

	Data:	%MW0.8196 = 4;
		%MW0.8197 = 5;
		%MW0.8198 = 6

	Source address at slave:	%MW0.8196 : 2004HEX = 8196DEC
	Target address at master:	awReadWord : ARRAY[0..2] OF WORD;
	The values of the flag words %MW0.8196..%MW0.8198 on the slave are written to the ARRAY awReadWord on the master.	

Modbus request of the master

Slave address	Function code	Slave operand address		Number of words		CRC	
		High	Low	High	Low	High	Low
01HEX	03HEX	20HEX	04HEX	00HEX	03HEX	4FHEX	CAHEX

Modbus response of the slave

Slave address	Function code	Number of bytes	Data	Data	Data	CRC	
			High / Low	High / Low	High / Low	High	Low
01HEX	03HEX	06HEX	00HEX /04HEX	00HEX /05HEX	00HEX /06HEX	40HEX	B6HEX

Parameterization of the COM_MOD_MAST block inputs
NB = Number of words

EN	COM	SLAVE	FCT	TIMEOUT	ADDR	NB	DATA
FALSE -> TRUE	1	1	3	Application-specific	8196	3	ADR (awReadWord[0])

FCT 3 or 4: Read n double words

The function code "read double word" is not defined in the Modbus RTU standard. This is why the double word is composed of a low word and a high word (depending on the manufacturer).

Master request

Slave address	Function code	Slave operand address		Number of words		CRC	
		High	Low	High	Low	High	Low

Slave response

Slave address	Function code	Number of bytes	...Data...	CRC	
				High	Low

Example:	Modbus interface of the master:	COM1
	Master reads from:	Slave 1
	Data:	%MD0.8193 = 32DEC = 00000020HEX; %MD0.8194 = 80000DEC = 00013880HEX
	Source address at slave:	%MD0.8193: 4002HEX = 16386DEC
	Target address at master:	adwReadDWord : ARRAY[0..1] OF DWORD
	The values of the flag double words %MD0.8193..%MD0.8194 on the slave are written to the ARRAY adwReadDWord on the master.	

Modbus request of the master

Slave address	Function code	Slave operand address		Number of words		CRC	
		High	Low	High	Low	High	Low
01HEX	03HEX	40HEX	02HEX	00HEX	04HEX	F0HEX	09HEX

Modbus response of the slave

Slave address	Function code	Number of bytes	Data	Data	Data	Data	CRC	
			High / Low	High / Low	High / Low	High / Low	High	Low
01HEX	03HEX	08HEX	00HEX / 00HEX	00HEX / 20HEX	00HEX / 01HEX	38HEX /80HEX	57HEX	B0HEX

Parameterization of the COM_MOD_MAST block inputs
NB = Number of words

EN	COM	SLAVE	FCT	TIMEOUT	ADDR	NB	DATA
FALSE -> TRUE	1	1	31	Application-specific	16386	4	ADR (adwReadDWord[0])

FCT 5: Write 1 bit

For the function code "write 1 bit", the value of the bit to be written is encoded in one word.

BIT = TRUE -> Data word = FF 00 HEX

BIT = FALSE -> Data word = 00 00 HEX

Master request

Slave address	Function code	Slave operand address		Number of words		CRC	
		High	Low	High	Low	High	Low

Slave address	Function code	Slave operand address		Data		CRC	
		High	Low	High	Low	High	Low

Slave response

Slave address	Function code	Slave operand address		Data		CRC	
		High	Low	High	Low	High	Low

Example:	Modbus interface of the master:	COM1
	Master writes to:	Slave 1
	Data:	bBit := TRUE
	Source address at master:	bBit : BOOL;
	Target address at slave:	%MX0.1026.7 : 2017HEX = 8215DEC
	The value of the BOOL variable bBit on the master is written to %MX0.1026.7 on the slave.	

Modbus request of the master

Slave address	Function code	Slave operand address		Data		CRC	
		High	Low	High	Low	High	Low
01HEX	05HEX	20HEX	17HEX	FFHEX	00HEX	37HEX	FEHEX

Modbus response of the slave (mirrored)

Slave address	Function code	Slave operand address		Data		CRC	
		High	Low	High	Low	High	Low
01HEX	05HEX	20HEX	17HEX	FFHEX	00HEX	37HEX	FEHEX

Parameterization of the COM_MOD_MAST block inputs
NB = Number of bits

EN	COM	SLAVE	FCT	TIMEOUT	ADDR	NB	DATA
FALSE -> TRUE	1	1	5	Application-specific	8215	1	ADR (bBit)

FCT 6: Write 1 word

Master request

Slave address	Function code	Slave operand address		Data		CRC	
		High	Low	High	Low	High	Low

Slave response

Slave address	Function code	Slave operand address		Data		CRC	
		High	Low	High	Low	High	Low

Example:	Modbus interface of the master:	COM1
	Master writes to:	Slave 1
	Data:	wData := 7
	Source address at master:	wData : WORD;
	Target address at slave:	%MW0.8199 : 2007HEX = 8199DEC
	The value of the WORD variable bBit on the master is written to %MW0.8199 on the slave.	

Modbus request of the master

Slave address	Function code	Slave operand address		Data		CRC	
		High	Low	High	Low	High	Low
01HEX	06HEX	20HEX	07HEX	00HEX	07HEX	72HEX	09HEX

Modbus response of the slave (mirrored)

Slave address	Function code	Slave operand address		Data		CRC	
		High	Low	High	Low	High	Low
01HEX	06HEX	20HEX	07HEX	00HEX	07HEX	72HEX	09HEX

Parameterization of the COM_MOD_MAST block inputs
NB = Number of words

EN	COM	SLAVE	FCT	TIMEOUT	ADDR	NB	DATA
FALSE -> TRUE	1	1	6	Application-specific	8215	1	ADR (wData)

FCT 7: Fast reading the status byte of the CPU

Master request

Slave address	Function code	CRC					
		High	Low				

Slave response

Slave address	Function code	Data byte	CRC			
			High	Low		

Example:	Modbus interface of the master:	COM1
	Master writes to:	Slave 1
	Data:	
	Source address at slave:	
	Target address at slave:	
	In version V1.x, this function always returns 0!	

Modbus request of the master

Slave address	Function code	CRC					
		High	Low				
01HEX	07HEX	41HEX	E2HEX				

Modbus response of the slave

Slave address	Function code	Data byte	CRC			
			High	Low		
01HEX	07HEX	00HEX	xxHEX	xxHEX		

Parameterization of the COM_MOD_MAST block inputs
NB = Number of bits

EN	COM	SLAVE	FCT	TIMEOUT	ADDR	NB	DATA
FALSE -> TRUE	1	1	7	Application-specific	0	0	ADR (BoolVar)

Note: In version V1.x, function 7 always returns 0!

FCT 15: Write n bits

Master request

Slave address	Function code	Slave operand address		Number of bits		Number of bytes	...Data...	CRC	
		High	Low	High	Low			High	Low

Slave response

Slave address	Function code	Slave operand address		Number of bits		CRC	
		High	Low	High	Low	High	Low

Example:	Modbus interface of the master:	COM1
	Master writes to:	Slave 1

	Data:	abWriteBool[0] := TRUE;
		abWriteBool[1] := FALSE;
		abWriteBool[2] := TRUE

	Source address at master:	abWriteBool : ARRAY[0..2] OF BOOL;
	Target address at slave:	%MX0.1026.1 : 2011HEX = 8209DEC
	The values of the BOOL variables abWriteBool[0]..abWriteBool[2] on the master are written to %MX0.1026.1..%MX0.1026.3 on the slave.	

Modbus request of the master

Slave address	Function code	Slave operand address		Number of bits		Number of bytes	Data	CRC	
		High	Low	High	Low			High	Low
01HEX	0FHEX	20HEX	11HEX	00HEX	03HEX	01HEX	05HEX	B4HEX	37HEX

Modbus response of the slave

Slave address	Function code	Slave operand address		Number of bits		CRC	
		High	Low	High	Low	High	Low
01HEX	0FHEX	20HEX	11HEX	00HEX	03HEX	4EHEX	0FHEX

Parameterization of the COM_MOD_MAST block inputs
NB = Number of bits

EN	COM	SLAVE	FCT	TIMEOUT	ADDR	NB	DATA
FALSE -> TRUE	1	1	15	Application-specific	8209	3	ADR (abWriteBool[0])

FCT 16: Write n words

Master request

Slave address	Function code	Slave operand address		Number of words		Number of bytes	...Data...	CRC	
		High	Low	High	Low			High	Low

Slave response

Slave address	Function code	Slave operand address		Number of words		CRC	
		High	Low	High	Low	High	Low

Example:	Modbus interface of the master:	COM1
	Master writes to:	Slave 1

	Data:	awWriteWord[0] := 1;
		awWriteWord[1] := 2;
		awWriteWord[2] := 3

	Source address at master:	awWriteWord : ARRAY[0..2] OF WORD;
	Target address at slave:	%MW0.8193 : 2001HEX = 8193DEC
	The values of the WORD variables awWriteWord[0]..awWriteWord[2] on the master are written to %MW0.8193..%MW0.8195 on the slave.	

Modbus request of the master

Slave address	Function code	Slave operand address	Number of words	Number of bytes	Data	Data	Data	CRC
		High / Low	High / Low		High / Low	High / Low	High / Low	High / Low
01HEX	10HEX	20HEX / 01HEX	00HEX / 03HEX	06HEX	00HEX / 01HEX	00HEX / 02HEX	00HEX / 03HEX	C0HEX / 84HEX

Modbus response of the slave

Slave address	Function code	Slave operand address		Number of words		CRC	
		High	Low	High	Low	High	Low
01HEX	10HEX	20HEX	01HEX	00HEX	03HEX	DAHEX	08HEX

Parameterization of the COM_MOD_MAST block inputs
NB = Number of words

EN	COM	SLAVE	FCT	TIMEOUT	ADDR	NB	DATA
FALSE -> TRUE	1	1	16	Application-specific	8193	3	ADR (awWriteWord[0])

FCT 16: Write n double words

The function code "write double word" is not defined in the Modbus RTU standard. This is why the double word is composed of a low word and a high word (depending on the manufacturer).

Master request

Slave address	Function code	Slave operand address		Number of words		Number of bytes	...Data...	CRC	
		High	Low	High	Low			High	Low

Slave response

Slave address	Function code	Slave operand address		Number of words		CRC	
		High	Low	High	Low	High	Low

Example:	Modbus interface of the master:	COM1
	Master writes to:	Slave 1
	Data:	adwWriteDWord[0] := 18DEC = 00000012HEX; adwWriteDWord[1] := 65561DEC = 00010019HEX;
	Source address at master:	adwWriteDWord : ARRAY[0..1] OF WORD;
	Target address at slave:	%MD0.8192 : 4000HEX = 16384DEC
	The values of the Double WORD variables adwWriteDWord[0].. adwWriteDWord[1] on the master are written to %MD0.8192..%MD0.8193 on the slave.	

Modbus request of the master

Slave address	Function code	Slave operand address	Number of words	Number of bytes	Data	Data	Data	Data	CRC
		High / Low	High / Low	High / Low	High / Low	High / Low	High / Low	High / Low	High / Low
01HEX	10HEX	40HEX / 00HEX	00HEX / 04HEX	00HEX / 08HEX	00HEX / 00HEX	00HEX / 12HEX	00HEX / 01HEX	00HEX / 19HEX	60HEX / B3HEX

Modbus response of the slave

Slave address	Function code	Slave operand address		Number of words		CRC	
		High	Low	High	Low	High	Low
01HEX	10HEX	40HEX	00HEX	00HEX	04HEX	DAHEX	0AHEX

Parameterization of the COM_MOD_MAST block inputs
NB = Number of words = 2 x Number of double words

EN	COM	SLAVE	FCT	TIMEOUT	ADDR	NB	DATA
FALSE -> TRUE	1	1	16	Application-specific	16384	4	ADR (adwWriteDWord[0])

Error telegram

In operating mode Modbus master, the AC500 does only send telegrams, if the parameters at the MODMAST inputs are logically correct. Nevertheless, it can happen that a slave cannot process the request of the master or that the slave cannot interpret the request due to transmission errors. In those cases, the slave returns an error telegram to the master. In order to identify this telegram as an error telegram, the function code returned by the slave is a logical OR interconnection of the function code received from the master and the value 80HEX.

Slave response

Slave address	Function code OR 80HEX	Error code	CRC	
			High	Low

Possible error codes of the slave

Code	Meaning
01DEC	The slave does not support the function requested by the master
02DEC	Invalid operand address in the slave
02DEC	Operand area exceeded
03DEC	At least one value is outside the permitted value range
12DEC	The amount of data is higher than the slave can process
13DEC	The telegram contains an odd number of words in case of double word access
10DEC	Length specifications in the telegram do not match
11DEC	The type of operand area and the function do not match

Example:

Modbus request of the master:			
	Function code:	01	(Read n bits)
	Slave operand address:	4000HEX = 16384DEC	(Area for read access disabled in slave)
Modbus response of the slave:			
	Function code:	81HEX	
	Error code:	03	

11.6 Function block COM_MOD_MAST

This function block is only required in the operating mode Modbus master. It handles the communication (transmission of telegrams to the slaves and reception of telegrams from the slaves). The function block can be used for the local interfaces COM1 and COM2 of the controller. A separate instance of the function block has to be used for each interface.

COM_MOD_MAST is contained in the library **Modbus_AC500_V1x.LIB (version V1.0 and later).**

Index - System Technology of the CPUs

System Description

AC500

Scalable PLC for
Individual Automation

System Technology
of the DC541-CM Module

DC541

Contents System Technology DC541-CM

The Interrupt and Counter Module DC541-CM

1 Functionality and configuration of the module DC541-CM

1.1 Functionality of the module DC541-CM

The interrupt and counter module DC541-CM has to be inserted into one of the coupler slots of the AC500. Slot 1 is the first slot on the left-hand side of the CPU. Depending on the used terminal base TB5x1, up to 4 DC541 modules can be used.

The module DC541-CM has 8 channels C0...C7 which can be configured individually as input or output as desired.

The device can be used in two **operating modes**: as interrupt input/output device (IO mode) or as counting device (counting mode).

In the operating mode **interrupt input/output**, the channels can be configured as follows:

- Input
- Output
- Interrupt input

In the operating mode **counting device**, the channels can be configured as follows:

- Input
- Output
- 32 bit up/down counter (uses C0...C3) (32 bit counter without limit)
- 32 bit periodic counter (limited 32 bit counter)
- Limiter for 32 bit counter (limit channel 0)
- 32 bit forward counter (count-up counter)
- Pulse width modulation (PWM)
- Time and frequency measurement
- Frequency output

Note: The implementation of the module functions of the DC541 is performed by means of function blocks. Access to the channels configured as normal inputs and outputs is also performed using a function block (DC541_IO).

The adjustment of the module's operating mode and its channel configuration is performed in the PLC configuration of the Control Builder. Specific function blocks are available for all module functions. The corresponding library **DC541_AC500_V11.LIB** is automatically included into the project.

The module's cycle time is set automatically depending on its channel configuration. The following values are possible for the cycle time:

I/O device:		50 µs
Counting device:	1-2 functions	50 µs
	3-4 functions	100 µs
	5-8 functions	200 µs

"**Functions**" are:

- PWM	- Pulse width modulator
- FREQ	- Time and frequency measurement
- FREQ_OUT	- Frequency output
- 32BIT_CNT	- 32 bit counter
- FWD_CNT	- 32 bit forward counter
- LIMIT	- Limit value monitoring for the 32 bit counter

The used cycle time can be read at output CYCLE of the block DC541_GET_CFG.

The following table shows an overview of all possible combinations.

Con-figu-red as	**Function/ can be con-figured for channel**	**C0**	**C1**	**C2**	**C3**	**C4 to C7**	**Max. number of channels for this function**	**Remark and reference to alternative combinations (a and b)**
Mode 1: Interrupt function; mutually exclusive with mode 2 (counting functions).								
Inter-rupt	Dig. input	1	1	1	1	4	8	Each channel can be configured individually as interrupt input or output.
	Interrupt inp.	1	1	1	1	4	8	
	Dig. output	1	1	1	1	4	8	
Mode 2: Counting functions and multifunctional I/Os; mutually exclusive with mode 1 (interrupt functions).								
Multi-function I/Os, functions digital I/Os, PWM, counters, time and frequency measuring	Dig. input	1	1	1	1	4	8	Normal input
	Dig. output	1	1	1	1	4	8	Normal output
	PWM, resolution 10 kHz	1	1	1	1	4	8	Outputs a pulsed signal with an adjustable on-off ratio.
	Frequency output, resolution 2.5 kHz	1	1	1	1	4	8	Outputs an adjustable frequency (endless output or output of a specified number of pulses).
	Up/down counter, 50 kHz	1	1	OK *1)	OK *1)	OK *1)	2	*1) a) Both channels (0 and 1) configured as 50 kHz counter => Channels 2 to 7 can be configured as digital I/Os. b) Only one channel (0 or 1) configured as 50 kHz counter => Second channel can be configured as counter < 50 kHz or for time/frequency measurement with a max. resolution of 200 µs. The remaining channels (2 to 7) can be configured as digital I/Os.
	Up/down counter, 5 kHz	1	1	1	1	OK *2)	4	*2) a) Four channels (0 to 3) configured as 5 kHz counter => Channels 4 to 7 can be configured as digital I/Os. b) Only a portion of the 4 channels (0 to 3) configured as 5 kHz counter => The other ones (of channels 0 to 3) can be configured as desired: as 2.5 kHz counter or for time/frequency measurement with a max. resolution of 200 µs or as digital I/Os. The remaining channels (4 to 7) can be configured as digital I/Os.
	Up/down counter, 2.5 kHz	1	1	1	1	4	8	

Con-figu-red as	Function/ can be con-figured for channel	C0	C1	C2	C3	C4 to C7	Max. number of channels for this function	Remark and reference to alternative combinations (a and b)
Mode 2: Counting functions and multifunctional I/Os; mutually exclusive with mode 1 (interrupt functions).								
Multi-function I/Os, functions digital I/Os, PWM, counters, time and frequency measuring	Time/ frequency measurement, resolution 50 µs	1	OK *3)	OK *3)	OK *3)	OK *3)	1	*3) Channel 0 configured for a max. resolution of 50 µs => Channels 1 to 7 can be configured as digital I/Os.
	Time/ frequency measurement, resolution 100 µs	1	1	OK *4)	OK *4)	OK *4)	2	*4) a) Two channels (0 and 1) configured for a max. resolution of 50 µs => Channels 3 to 7 can be configured as digital I/Os. b) Only one channel (0 or 1) configured for a max. resolution of 50 µs => Second channel (0 or 1) can be configured as counter < 50 kHz or for time/frequency measurement with a max. resolution of 200 µs. The remaining channels (2 to 7) can be configured as digital I/Os.
	Time/ frequency measurement, resolution 200 µs	1	1	1	1	4	8	Times, frequencies and rotational speeds are measured with a max. resolution of 200 µs.
High-speed counters	Bidirectional 32 bit counter, 50 kHz max.	Channels 0 to 3: Track A, track B, zero track, touch trigger				OK *6)	1	For connection of an incremental transmitter. For signals up to 50 kHz. This frequency corresponds to a motor with a rotational speed of 3000 rpm. The counter always uses the first 4 channels (0 to 3). *6) The remaining channels (4 to 7) can be configured as limit values, as 5 kHz counters, for time/frequency measurement with a resolution of 200 µs or as digital I/Os
	Axis of rotation (endless counting)	1				OK *7)	1	"Endless" forward counting. An overflow occurs corresponding to the 32 bit value. *7) The remaining channels can be configured as limit values, as 5 kHz counters, for time/frequency measurement with a resolution of 200 µs or as digital I/Os.
	32 bit counter incl. sign	1				OK *8)	1	*8) The remaining channels can be configured as limit values, as 5 kHz counters, for time/frequency measurement with a resolution of 200 µs or as digital I/Os.
	Limit values for 32 bit counter	OK *9)				1	1	Various counting values of the 32 bit counter can be displayed directly via these outputs. *9) In this case, the channels 0 to 3 are used as 32 bit counters.

1.2 Application examples for the module DC541-CM

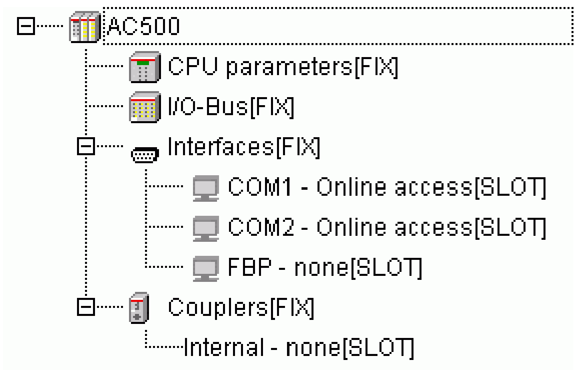
The module DC541 can be used for a wide variety of control tasks. In this documentation, a small choice of possible tasks is described together with the corresponding configuration for the DC541 and with reference to the corresponding example projects.

1.3 Configuring the module DC541-CM

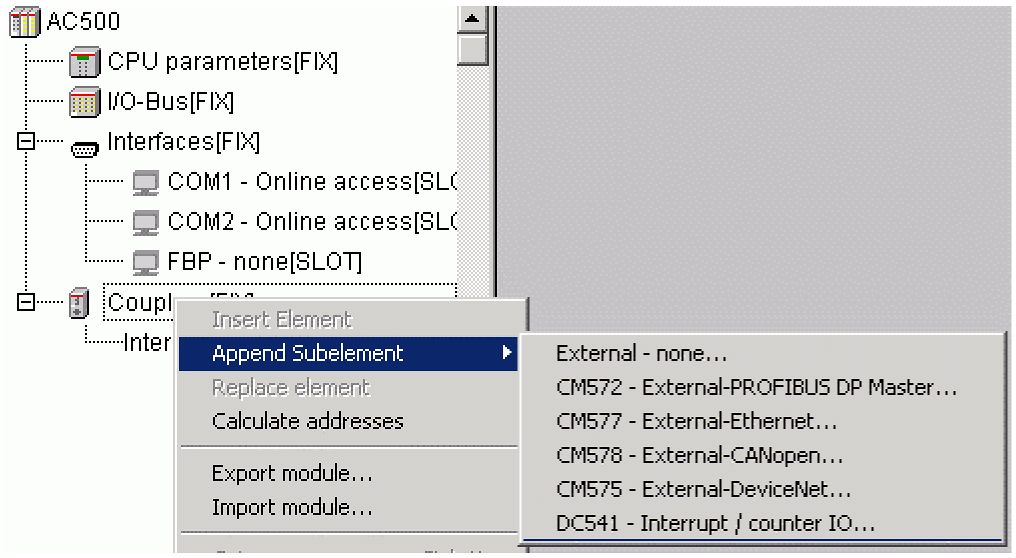
The configuration of the interrupt and counting module DC541 is done in the PLC configuration of the Control Builder PS501. A detailed description of the PLC configuration for the AC500 can be found in chapter "System Technology of the CPUs" / "PLC configuration".

This section describes the procedure for configuring the DC541 when used with a AC500 CPU (PM581).

After creating a project for the target system PM581, the PLC configuration looks as follows:

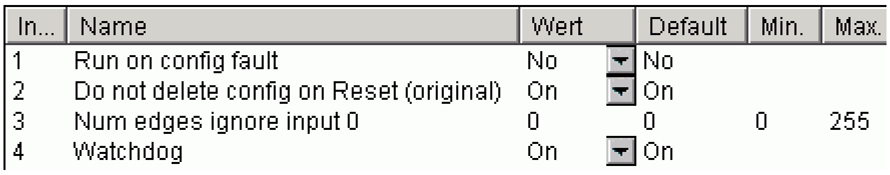


The device DC541 has to be entered at the corresponding coupler slot according to the hardware setup. Right click the entry "Couplers[FIX]" to open the context menu for appending a coupler. Select "Append Subelement" to display all devices available for the coupler slots.



If the DC541 is inserted in slot 1 (i.e. the first slot on the left of the CPU), you then have to select the entry "DC541 - Interrupt / counter IO". If the DC541 is e.g. inserted in slot 2 and slot 1 is empty or should be used for another coupler, you first have to enter an "External - none" for an empty slot or the inserted coupler respectively. In the example, the coupler is inserted in slot 1.

After the device has been added successfully, the module parameters have to be specified:



In...	Name	Wert	Default	Min.	Max.
1	Run on config fault	No	No		
2	Do not delete config on Reset (original)	On	On		
3	Num edges ignore input 0	0	0	0	255
4	Watchdog	On	On		

The following module parameters are available:

Parameter	Default value	Value	Meaning
Run on config fault	No	No	In case of a configuration error, the user program is not started.
		Yes	The user program is started even if the internal Ethernet coupler is configured incorrectly.
Do not delete config on Reset (original)	On	No	The configuration of the DC541 is **not** deleted in case of a reset (original).
		Yes	In case of a reset (original), the configuration of the DC541 is deleted, too.
Num edges ignore input 0	0	0...255	Number of edges that may occur at input 0 without initiating the interrupt task, if channel C0 is configured as interrupt input.
Watchdog	On	On	Mutual time monitoring between the CPU and the DC541 is switched on.
		Off	No time monitoring.

The library DC541_AC500_V11.LIB is included automatically during the first compilation of the project after the device has been added to the PLC configuration.

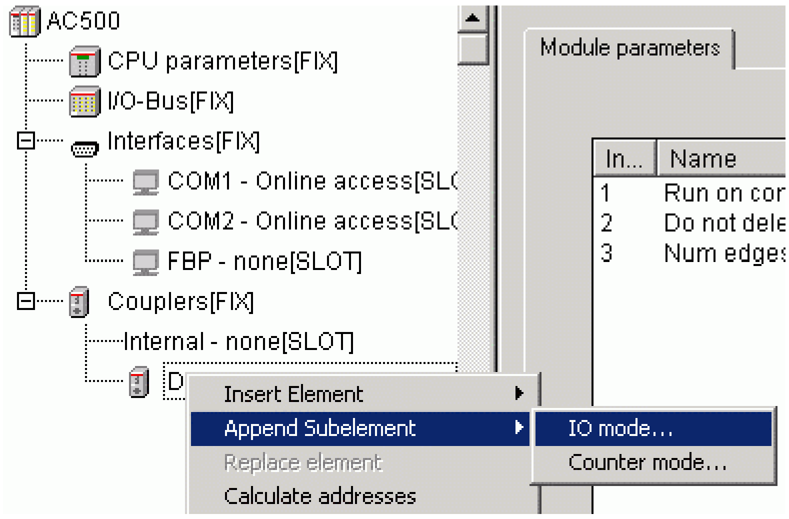
2 Module used as interrupt I/O device

2.1 Configuring the module DC541-CM for use as interrupt I/O device

For use as an interrupt I/O device, the device and the channels have to be configured in the PLC configuration accordingly.

How to add the device in the PLC configuration is described in the chapter before.

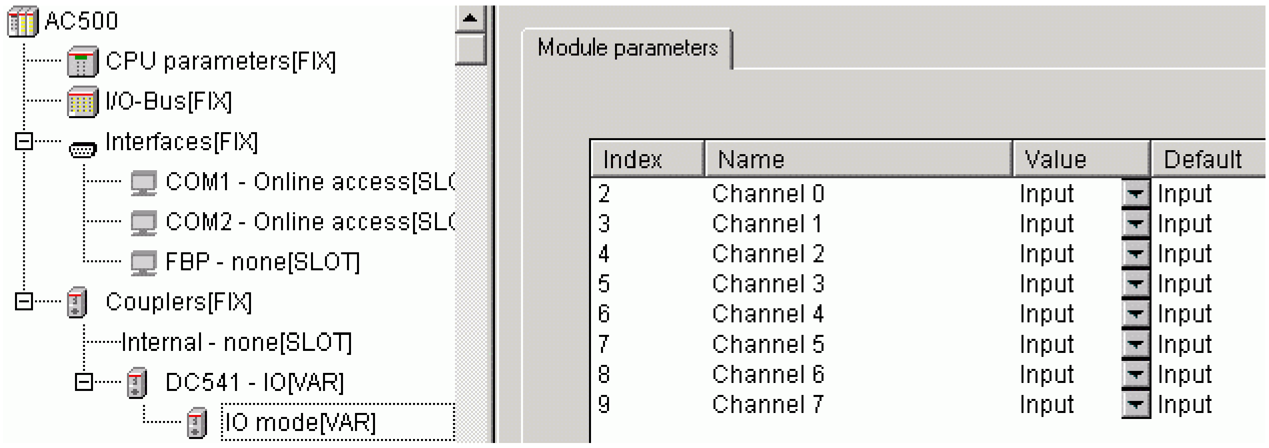
The next step is to set the operating mode of the device. To do so, position the cursor on the entry of the device DC541, right click to open the context menu and then select the function "Append Subelement":



In...	Name
1	Run on cor
2	Do not dele
3	Num edge

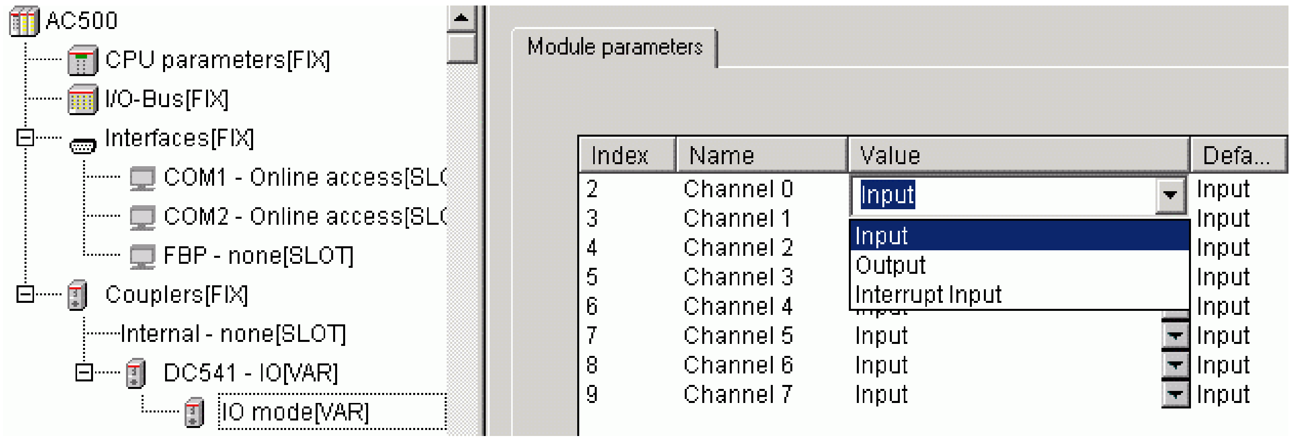
Then, select "IO mode" to configure the device as an interrupt IO.

In the module parameters, you can specify the channels C0...C7 as inputs, outputs or interrupt inputs.



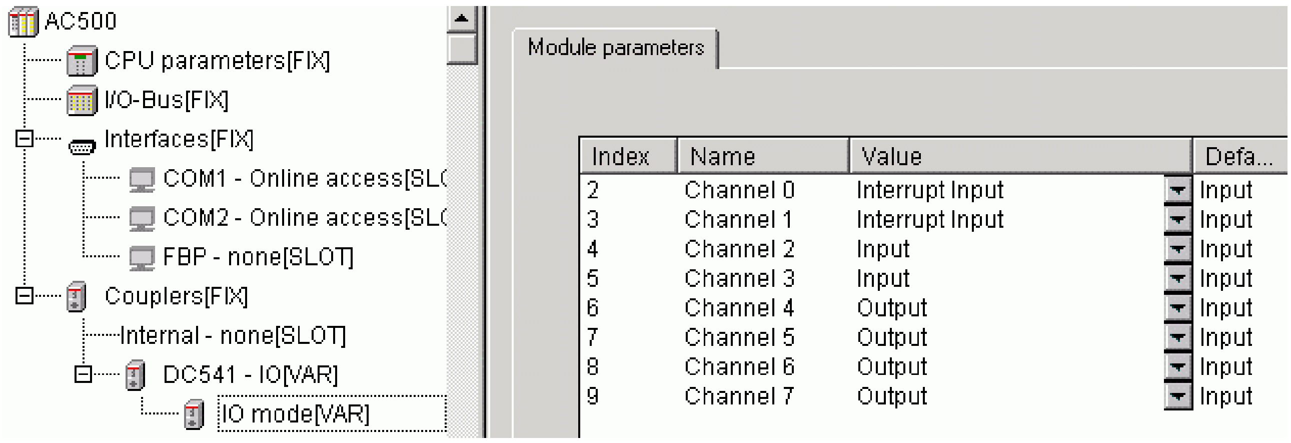
Index	Name	Value	Default
2	Channel 0	Input	Input
3	Channel 1	Input	Input
4	Channel 2	Input	Input
5	Channel 3	Input	Input
6	Channel 4	Input	Input
7	Channel 5	Input	Input
8	Channel 6	Input	Input
9	Channel 7	Input	Input

To do so, select the corresponding value for each channel.



Index	Name	Value	Defa...
2	Channel 0	Input	Input
3	Channel 1		Input
4	Channel 2		Input
5	Channel 3		Input
6	Channel 4		Input
7	Channel 5	Input	Input
8	Channel 6	Input	Input
9	Channel 7	Input	Input

In the example, the channels 0 and 1 are configured as interrupt inputs, channels 2 and 3 as inputs and channels 4 to 7 as outputs. In this case, the configuration looks as follows:



Index	Name	Value	Defa...
2	Channel 0	Interrupt Input	Input
3	Channel 1	Interrupt Input	Input
4	Channel 2	Input	Input
5	Channel 3	Input	Input
6	Channel 4	Output	Input
7	Channel 5	Output	Input
8	Channel 6	Output	Input
9	Channel 7	Output	Input

The configuration of the device DC541 for use as interrupt IO device is now completed.

The specified configuration can be read using the block DC541_GET_CFG.

2.2 Creating an interrupt task for the interrupt inputs

If one or more channels of the DC541 are configured as interrupt inputs, a corresponding interrupt task has to be created to enable the processing of the interrupt(s).

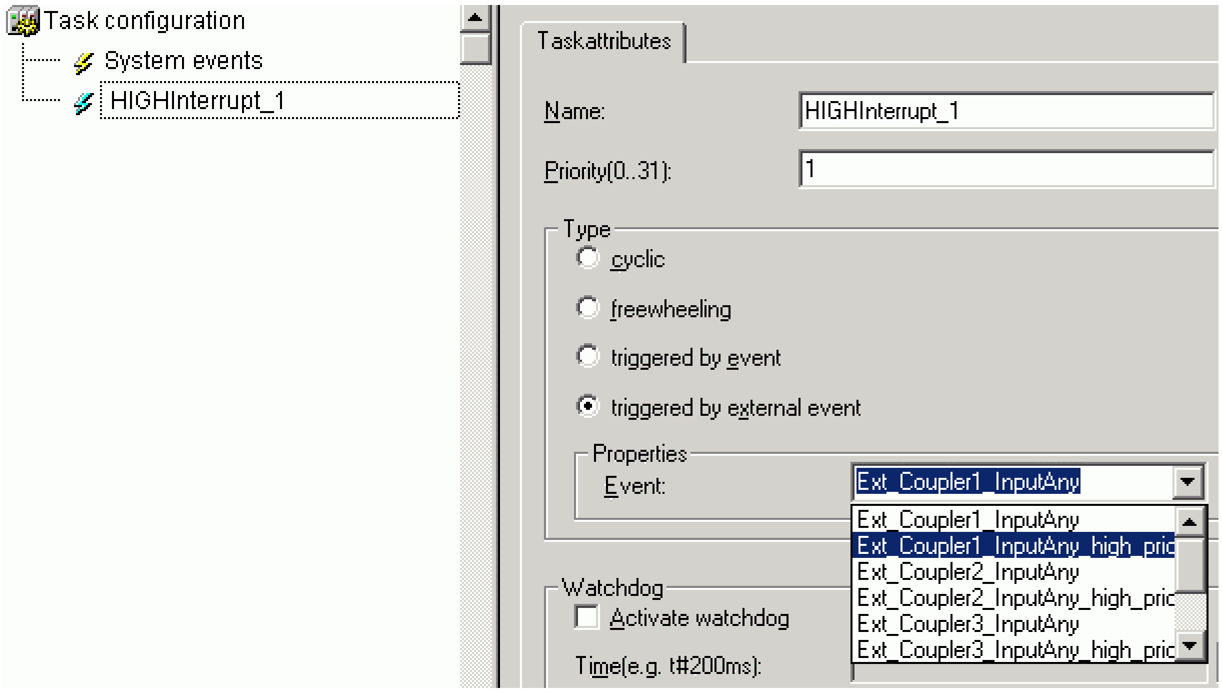
For this purpose, a new task has to be added in the task configuration of the Control Builder. Enter the task name, set the task type to "external event triggered" and specify the event that triggers the task.

For each coupler slot, two types of interrupt tasks are available in the "Event" list box:

- **Ext_CouplerX_InputAny:**
  The task is triggered by any interrupt from coupler slot X with the priority specified in the Priority field (0...31).
- **Ext_CouplerX_InpuAny_high_prio:**
  The task is triggered by any interrupt from coupler slot X with highest priority, i.e. with a priority higher than the max. adjustable "0" and higher than the priority of the communication task. In this case, the priority (0...31) specified in the Priority field is without any significance.

⚠ **CAUTION: If the interrupt task is started with high priority (Ext_CouplerX_InpuAny_high_prio), the program execution time must not be longer than approx. 400 µs. Otherwise online access is no longer possible.**

In the example below, the task is named HIGHInterrupt_1, meaning that it is a high-priority interrupt from coupler slot 1. The task type is "external event triggered" and the event to trigger the task is "Ext_Coupler1_InputAny_high_priority".


Task configuration
System events
HIGHInterrupt_1

Taskattributes

Name: HIGHInterrupt_1

Priority(0..31): 1

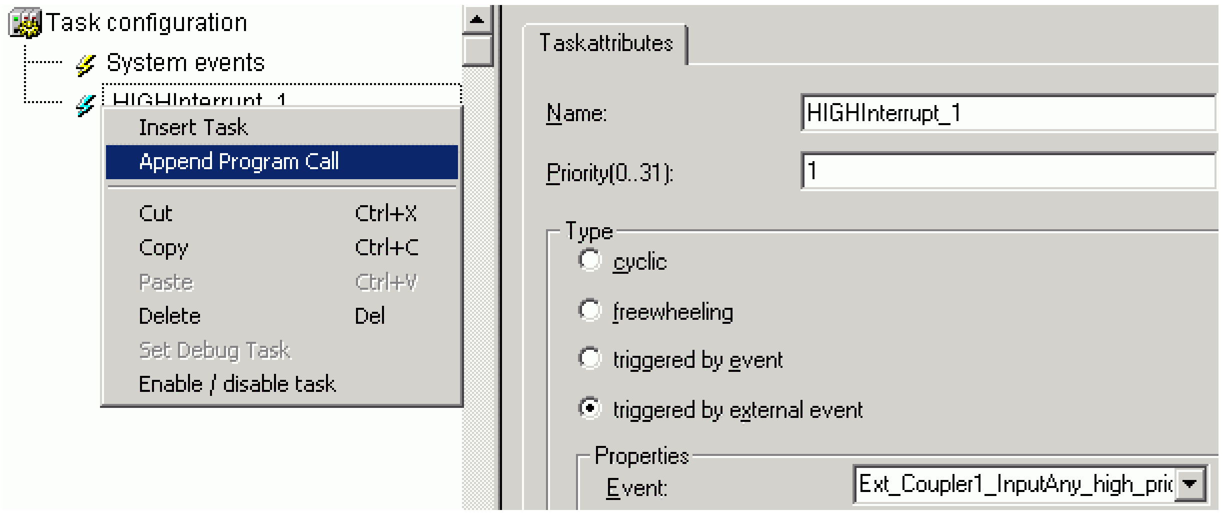
Type
( ) cyclic
( ) freewheeling
( ) triggered by event
(●) triggered by external event

Properties
Event: Ext_Coupler1_InputAny

Ext_Coupler1_InputAny
Ext_Coupler1_InputAny_high_pric
Ext_Coupler2_InputAny
Ext_Coupler2_InputAny_high_pric
Ext_Coupler3_InputAny
Ext_Coupler3_InputAny_high_pric

Watchdog
[ ] Activate watchdog
Time(e.g. t#200ms):

Like for all other tasks, a program call has to be assigned to the task.



Taskattributes

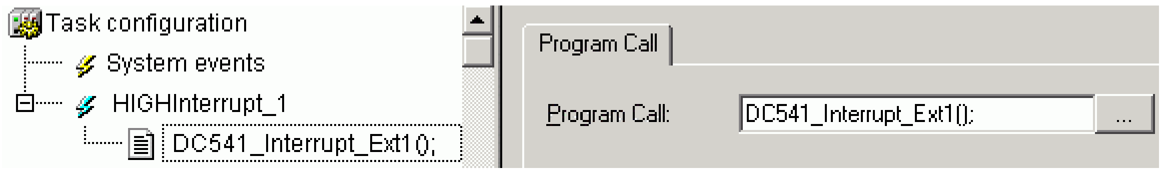
Name: HIGHInterrupt_1

Priority(0..31): 1

Type
( ) cyclic
( ) freewheeling
( ) triggered by event
(●) triggered by external event

Properties
Event: Ext_Coupler1_InputAny_high_pric

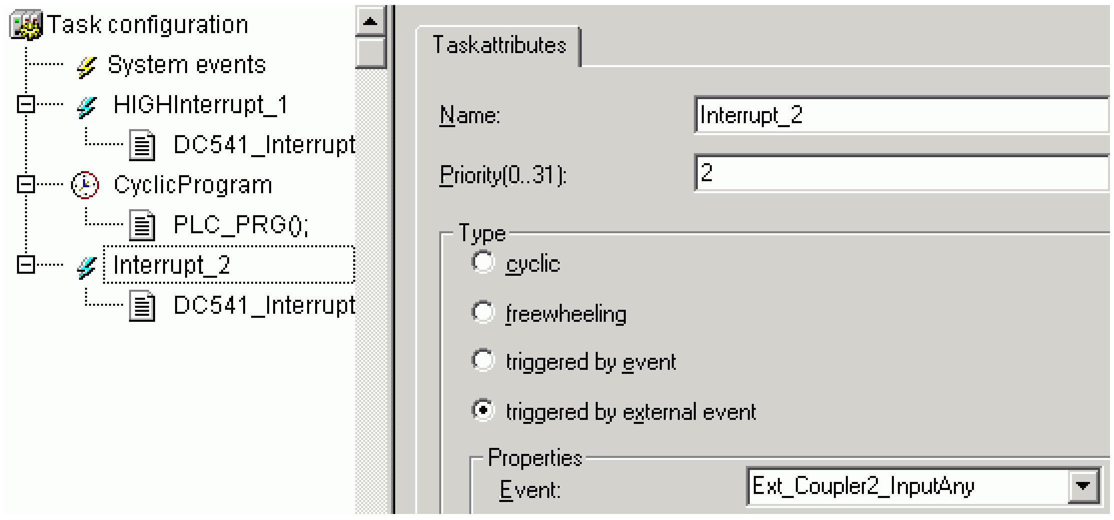
In the example, the program DC541_Interrupt_Ext1() shall be started with any interrupt from coupler slot 1.


Task configuration
System events
HIGHInterrupt_1
DC541_Interrupt_Ext1();

Program Call

Program Call: DC541_Interrupt_Ext1(); ...

The task configuration for an AC500 equipped with two DC541 modules inserted in the coupler slots 1 and 2 and containing one cyclically running "background program" PLC_PRG could for example look as follows. Here, an interrupt from slot 1 should start the program DC541_Interrupt_Ext1 with high priority, an interrupt from slot 2 should start the program DC541_Interrupt_Ext2 with priority 2:


Task configuration
System events
HIGHInterrupt_1
DC541_Interrupt
CyclicProgram
PLC_PRG();
Interrupt_2
DC541_Interrupt

Taskattributes

Name: Interrupt_2

Priority(0..31): 2

Type
( ) cyclic
( ) freewheeling
( ) triggered by event
(●) triggered by external event

Properties
Event: Ext_Coupler2_InputAny

2.3 Structure of the interrupt program

The following blocks contained in the library DC541_AC500_V11.LIB are available for the interrupt program:

- DC541_INT_IN Determination of the interrupt initiating source
- DC541_IO Reading and writing of channels C0...C7

It is possible to start one interrupt task per coupler slot. This task can be started by any channel (C0...C7) configured as interrupt input. Therefore, it is necessary for the interrupt program to differentiate which channel(s) triggered the interrupt in order to enable the processing of the corresponding actions.

The information whether a channel (C0...C7) has triggered an interrupt since the last call of the block is provided by the outputs IN0...IN7 of the block DC541_INT_IN. This is why this block always has to be called at the beginning of the interrupt program, if more than one channel is configured as interrupt input.

The access to the channels configured as inputs or outputs is done using the block DC541_IO. Therefore, it makes sense to call this block at the beginning of the interrupt program in order to read the inputs and at the end of the interrupt program in order to write the outputs.

2.4 Configuration example: DC541-CM used as interrupt I/O device

The configuration example described in this section is contained in the following folder on the Control Builder PS501 CD-ROM (V1.1 and later):

..\CD_AC500\Examples\DC541

File name:

DC541_DokuInterruptExample_PM591_V11.pro

Hardware configuration:

The example control system shall have the following configuration:

- Terminal base TB521 (two coupler slots)
- DC541 in coupler slot 1 (first slot on the left of the CPU)
- PM591-ETH CPU with internal Ethernet coupler (configuration using SYCON.net)
- I/O module DC532 on the I/O bus

Wiring:

The channels are connected as follows:

DC532 / C16 -------------- DC541 / C0
DC532 / C17 -------------- DC541 / C1
DC532 / C18 -------------- DC541 / C2
DC532 / C19 -------------- DC541 / C3
DC532 / C20 -------------- DC541 / C4
DC532 / C21 -------------- DC541 / C5

PLC configuration:

- DC541 in slot 1, operating mode "IO mode"

- Configuration:	Channels	C0...C4	Interrupt input
	Channel	C5	Input
	Channels	C6...C7	Outputs

- Specification of the Ethernet coupler as internal coupler (if available)
(Ethernet coupler configuration using SYCON.net)

- DC532 on the I/O bus

Task configuration:

- Task 1: Cyclic program / Prio = 10 / Interval = t#10ms / PLC_PRG
- Task 2: HIGHInterrupt_1 / DC541_Interrupt_Ext1()

Purpose of the interrupt program DC541_Interrupt_Ext1():

The interrupt program should fulfill the following functionality:

- Counting of all interrupts
- Counting of the interrupts per input
- Calculation of the interrupt frequency in [Int/s]
- Reporting of the number of interrupts per input
- Input C4: Resetting the counters
- Input C5: Input
- Output C6: Status of input C5
- Output C7: Toggle output

The **declaration part** of the program looks as follows:

```
PROGRAM DC541_Interrupt_Ext1
VAR
    dwIntCount          : DWORD;                        (* count all interrupts *)
    dwIntCountOld       : DWORD;                        (* start value for next measure *)
    tActual             : TIME;                         (* systemtick in ms *)
    tStart              : TIME;                         (* start value of systemtick for next
                                                        calculation *)
    dwUsedTime          : DWORD;                        (* time for 1000 interrupts in ms *)
    dwFrequenz          : DWORD;                        (* interrupt frequency in [Int / sec] *)
    DC541_IntSource     : DC541_INT_IN;                 (* instance FB: read interrupt source *)
    DC541_Ios           : DC541_IO;                     (* instance FB: read/write inputs/outputs *)
    dwCount_InX         : ARRAY[0..cbyDC541_IntInp] OF  (* count interrupts of In0..In3 *)
                        DWORD;
    dwCount_InXOld      : ARRAY[0..cbyDC541_IntInp] OF  (* start value for next 1000 interrupts *)
                        DWORD;
    dwIntHisto          : ARRAY[0..cbyDC541_IntInp,     (* histo data C0...C3 *)
                        0..cbyDC541_MaxHist] OF DWORD;
    wIndex              : WORD;                         (* index for histo data *)
    byInd               : BYTE;                         (* loop index *)
END_VAR
VAR CONSTANT
    cbyDC541_SLOT       : BYTE := 1;                    (* SLOT number of DC541 *)
    cbyDC541_MaxHist    : BYTE := 9;                    (* max number of histo entries *)
    cbyDC541_IntInp     : BYTE := 4;                    (* number of interrupt inputs -1 *)
END_VAR
```

The **instruction part** looks as follows:

At the beginning, the interrupts are counted in dwIntCount. After each 1000 interrupts, a calculation of the frequency is performed and the counting values for the interrupts per input are stored.

```
dwIntCount := dwIntCount + 1;                               (* count all interrupts *)
IF dwIntCount - dwIntCountOld >= 1000 THEN                  (* after 1000 interrupts -> calculate frequency  *)
    dwIntCountOld := dwIntCount;                            (* save dwIntCount for next call *)
    tActual := TIME();
    dwUsedTime := TIME_TO_DWORD(tActual - tStart);          (* duration in ms for 1000 interrupts *)
    dwFrequenz := 1000000 / dwUsedTime;                     (* [Interrupt / sec] 1000 Int * 1000 ms/sec *)
    tStart := tActual;                                      (* for next measure *)
    dwIntHisto[0,wIndex] := dwCount_InX[0] - dwCount_InXOld[0];   (* IN0 interrupts of last 1000 *)
    dwCount_InXOld[0] :=dwCount_InX[0];                     (* start value for next measure *)
    dwIntHisto[1,wIndex] := dwCount_InX[1] - dwCount_InXOld[1];   (* IN1 interrupts of last 1000 *)
    dwCount_InXOld[1] :=dwCount_InX[1];                     (* start value for next measure *)
    dwIntHisto[2,wIndex] := dwCount_InX[2] - dwCount_InXOld[2];   (* IN2 interrupts of last 1000 *)
    dwCount_InXOld[2] :=dwCount_InX[2];                     (* start value for next measure *)
    dwIntHisto[3,wIndex] := dwCount_InX[3] - dwCount_InXOld[3];   (* IN3 interrupts of last 1000 *)
    dwCount_InXOld[3] :=dwCount_InX[3];                     (* start value for next measure *)
    wIndex := wIndex + 1;                                   (* increase index *)
    IF wIndex > cbyDC541_MaxHist THEN wIndex := 0; END_IF;  (* reset index, if >1000 *)
END_IF; (* 1000 Interrupts *)
```

After this, the block DC541_INT_IN is called to identify the interrupt source and then the interrupt counters of the channels are updated depending on the outputs of this block.

```
(* Read interrupt source --> if output  = TRUE --> interrupt since last call *)
DC541_IntSource(EN := TRUE, SLOT := cbyDC541_SLOT);

(* count the interrupts for each interrupt input C0..C3 *)
dwCount_InX[0] := dwCount_InX[0] + BOOL_TO_DWORD(DC541_IntSource.IN0);
dwCount_InX[1] := dwCount_InX[1] + BOOL_TO_DWORD(DC541_IntSource.IN1);
dwCount_InX[2] := dwCount_InX[2] + BOOL_TO_DWORD(DC541_IntSource.IN2);
dwCount_InX[3] := dwCount_InX[3] + BOOL_TO_DWORD(DC541_IntSource.IN3);
dwCount_InX[4] := dwCount_InX[4] + BOOL_TO_DWORD(DC541_IntSource.IN4);
```

In case of an interrupt on channel 4, the counters are reset.

```
IF DC541_IntSource.IN4 THEN                         (* Input channel C4 = TRUE *)
    dwIntCount := dwIntCountOld := 0;               (* reset count all interrupts *)
    FOR byInd := 0 TO cbyDC541_IntInp-1 DO          (* reset channel interrupt counters C0..C3 *)
    dwCount_InX[byInd] := dwCount_InXOld[byInd] := 0;
    END_FOR; (* byInd *)
    wIndex := 0;                                    (* start historical data from 0 *)
END_IF; (* C4 = TRUE *)
```

At the end, the static inputs and outputs are processed, i.e.:

- reading the inputs,
- execution of actions
- writing the outputs.

```
(* Read inputs of DC541 *)
DC541_IOs(EN := TRUE, SLOT := cbyDC541_SLOT );


DC541_IOs.OUT6 := DC541_IOs.IN5;                  (* C6 := state of input channel C5 *)
DC541_IOs.OUT7 := NOT DC541_IOs.OUT7;             (* toggle channel C7 *)


(* Write outputs to DC541*)
DC541_IOs(EN := TRUE, SLOT := cbyDC541_SLOT);
```

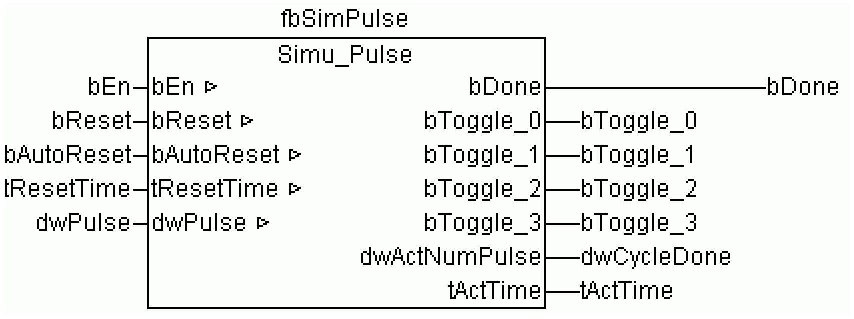
Purpose of the cyclic program PLC_PRG:

The cyclic program PLC_PRG contains the following functions:

- Cycles counter
  dwC := dwC + 1;
- Reading the configuration of the DC541
  Calling of block DC541_GET_CFG
- Reading the status of the DC541
  Calling of block DC541_STATE
- Reading/writing the static channels of the DC541
  Calling of block DC541_IO
- Simulation of the interrupts for the DC541
  Calling of block Simu_Pulse

The blocks DC541_GET_CFG, DC541_STATE and DC541_IO are contained in the library DC541_AC500_V11.lib and described in detail in the library documentation.

The block Simu_Pulse is used to generate an adjustable number of pulses. Its representation in the function block diagram (FBD) is as follows:



The meanings of the block's inputs and outputs are as follows:

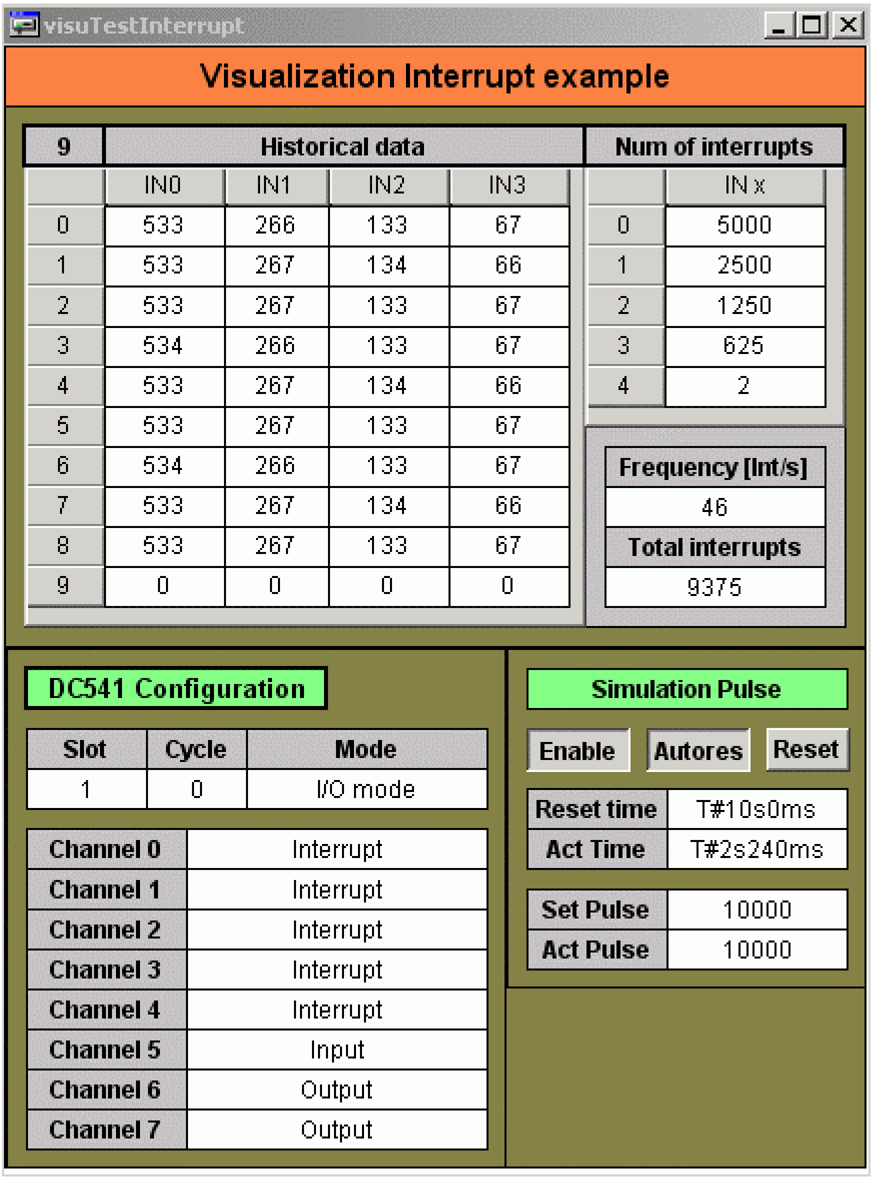
Instance		fbSimuPulse	Instance name
bEn	Input/Output	BOOL	Enabling of the pulse output
bAutoReset	Input/Output	BOOL	Automatic reset of the pulse counter after the specified number of pulses have been output and after expiration of tResetTime
bReset	Input/Output	BOOL	Reset of the pulse counter
tResetTime	Input/Output	TIME	Time until the reset is initiated after the specified number of pulses is reached, if bAutoReset = TRUE
dwPulse	Input/Output	DWORD	Number of pulses to be output: =0: Endless mode (pulse output continues until bEn = FALSE or bReset = TRUE > 0: Cyclic mode (output of the specified number of pulses)
bDone	Output	BOOL	Completion message after tResetTime has expired or bReset = TRUE for 1 cycle
bToggle_0	Output	BOOL	Provides a FALSE->TRUE edge with each 2nd call (i.e. the output is toggled with each call)
bToggle_1	Output	BOOL	Provides a FALSE->TRUE edge with each 4th call
bToggle_2	Output	BOOL	Provides a FALSE->TRUE edge with each 8th call
bToggle_3	Output	BOOL	Provides a FALSE->TRUE edge with each 16th call
dwActPulse	Output	DWORD	Displays the number of pulses output (corresponds to the number of edges at bToggle_0)
tActTime	Output	TIME	Displays the elapsed time while tResetTime is running

In the example, bEn: = bAutoReset: = TRUE. 10000 pulses are output (dwSetPulse). After the specified number of pulses has been reached, a wait time of 10 seconds is applied and then counting is started from the beginning.

The example has a visualization implemented which can be used to operate the program. After 10000 pulses, the visualization looks as follows:

9375 interrupts are generated:
  5000 x C0 + 2500 x C1 + 1250 x C2 + 625 x C3 = 9375

Act Pulse					Triggers the following interrupts:
Value	**IN 3 8**	**IN 2 4**	**IN 1 2**	**IN 0 1**	
0	0	0	0	0	none
1	0	0	0	1	IN 0 -> in every 2. cycle (10000 : 2 = 5000)
2	0	0	1	0	IN 1 -> in every 4. cycle (10000 : 4 = 2500)
3	0	0	1	1	IN 0
4	0	1	0	0	IN 2 -> in every 8. cycle (10000 : 8 = 1250)
5	0	1	0	1	IN 0
6	0	1	1	0	IN 1
7	0	1	1	1	IN 0
8	1	0	0	0	IN 3 -> in every 16. cycle (10000 : 16 = 625)
9	1	0	0	1	IN 0
10	1	0	1	0	IN 1
11	1	0	1	1	IN 0
12	1	1	0	0	IN 2
13	1	1	0	1	IN 0
14	1	1	1	0	IN 1
15	1	1	1	1	IN 0
16	0	0	0	0	none



Visualization Interrupt example

9	Historical data			
	IN0	IN1	IN2	IN3
0	533	266	133	67
1	533	267	134	66
2	533	267	133	67
3	534	266	133	67
4	533	267	134	66
5	533	267	133	67
6	534	266	133	67
7	533	267	134	66
8	533	267	133	67
9	0	0	0	0

Num of interrupts	
	IN x
0	5000
1	2500
2	1250
3	625
4	2

Frequency [Int/s]
46
Total interrupts
9375

DC541 Configuration

Slot	Cycle	Mode
1	0	I/O mode

Channel 0	Interrupt
Channel 1	Interrupt
Channel 2	Interrupt
Channel 3	Interrupt
Channel 4	Interrupt
Channel 5	Input
Channel 6	Output
Channel 7	Output

Simulation Pulse

Reset time	T#10s0ms
Act Time	T#2s240ms

Set Pulse	10000
Act Pulse	10000

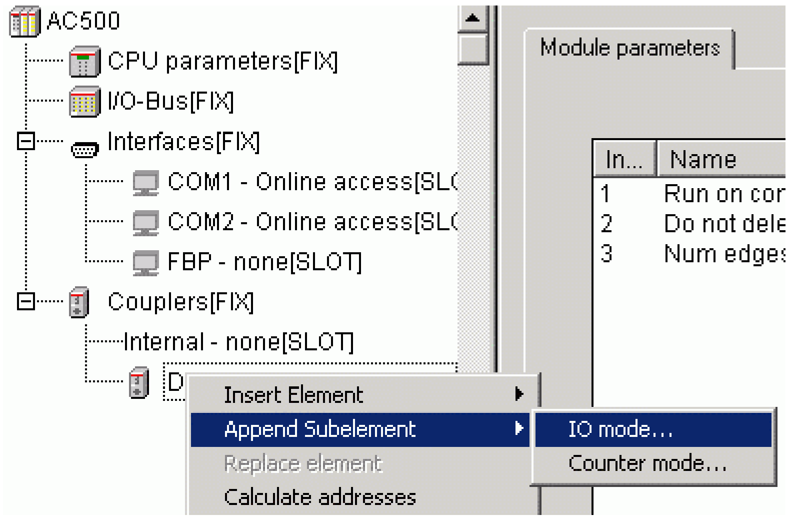
3 Module used as counting device

3.1 Configuring the module DC541-CM for use as counting device

For use as a counting device, the device and the channels have to be configured in the PLC configuration accordingly.

How to add the device in the PLC configuration is described in the chapter before (link to "Configuring the module DC541").

The next step is to set the operating mode of the device. To do so, position the cursor on the entry of the device DC541, right click to open the context menu and then select the function "Append Subelement":

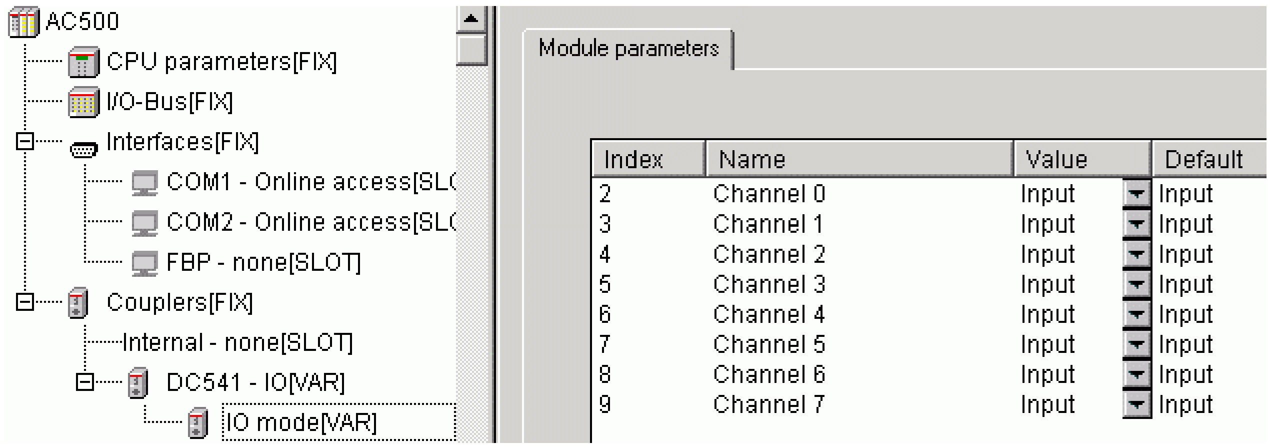


In...	Name
1	Run on cor
2	Do not dele
3	Num edges

Then, select "Counter mode" to configure the device as a counting device.

In the module parameters, the channels C0...C7 can be configured as:

Input	- Input
Output	- Output
32 bit counter	- 32 bit up/down counter (uses channels C0...C3)
Limit channel 0	- Limiter for 32 bit counter
Forward counter	- 32 bit forward counter
PWM	- Pulse width modulation
Frequency measurement	- Time and frequency measurement
Frequency output	- Frequency output



Index	Name	Value	Default
2	Channel 0	Input	Input
3	Channel 1	Input	Input
4	Channel 2	Input	Input
5	Channel 3	Input	Input
6	Channel 4	Input	Input
7	Channel 5	Input	Input
8	Channel 6	Input	Input
9	Channel 7	Input	Input

To do so, select the corresponding value for each channel.

Note: If channel C0 is configured as "32 bit counter", the channels C1...C3 are automatically used, too. In the PLC configuration, these channels are left at the default setting "Input".

The specified configuration can be read using the block DC541_GET_CFG.

3.2 Calling the counting functions of the DC541-CM

Function blocks are available for all module functions of the DC541 used as counting device. The blocks are contained in the library DC541_AC500_V11.lib and described in detail in the library documentation.

The table below lists the assignment of the individual functions to the available function blocks:

Function	Configuration	Function block
Input	Input	DC541_IO
Output	Output	DC541_IO
32 bit up/count-down counter	32 bit up/down counter	DC541_32BIT_CNT
Limiter for the 32 bit counter	Limit channel 0	DC541_LIMIT
32 bit forward counter	32 bit forward counter	DC541_FWD_CNT
Pulse width modulation	PWM	DC541_PWM
Time and frequency measurement	Time and frequency measurement	DC541_FREQ
Frequency output	Frequency output	DC541_FREQ_OUT

The blocks **DC541_GET_CFG** and **DC541_STATE** are available for diagnosis.

The following section describes the individual functions and provides a configuration example for each function.

3.3 The 32 bit up/down counter of module DC541-CM

3.3.1 Description of the module's up/down counter functionality

The 32 bit up/down counter functionality is provided by the block **DC541_32BIT_CNT**.

This 32 bit counter is a count-up/count-down counter with a directional discriminator. The counter can be used in two **counting modes**:

- **EN_UD = FALSE: Encoder mode**

  Connection of an incremental transmitter (track A / track B, offset by 90°)

  It is possible to count signals up to approx. 60 kHz. This corresponds to a motor with a rotational speed of 3600 rpm and a transmitter with 1000 pulses per rotation. Pulse multiplication (x2 or x4) is not used.

- **EN_UD = TRUE: Up / down mode**

  Up-/down counter

  It is possible to count signals up to approx. 60 kHz. Count-up for signals on channel C1, count-down for signals on channel C0.

The counter always uses the channels C0...C3 of the DC541:

- C0: Track A of the incremental transmitter
- C1: Track B of the incremental transmitter
- C2 and C3: Reference cam or touch trigger

The counter can be used in two **operating modes**:

- Infinite counter (endless mode)
- Limiting counter (limit mode)

The operating mode is selected at input EN_LIM.

If EN_LIM = FALSE, the counter operates as infinite counter (endless mode). An overflow occurs corresponding to the 32 bit value at 16#FFFFFFFF = 4 294 967 295. In this mode, any exceeding of the limit value LIM_MAX or falling below the limit value LIM_MIN is displayed at the outputs MAX_LIM or MIN_LIM.

If EN_LIM = TRUE (limit mode), the counting range is between the limit values LIM_MIN and LIM_MAX. In case of an overflow, i.e. if LIM_MAX is reached, the counter restarts again at LIM_MIN.

The upper limit value LIM_MAX has to be higher than the lower limit value LIM_MIN. If the lower limit value LIM_MIN is higher than the upper limit value LIM_MAX, a corresponding error message is applied at the outputs ERR/ERNO. In this case, the values for LIM_MIN and LIM_MAX are not forwarded to the DC541. The difference between LIM_MAX and LIM_MIN has to be at least twice the number (frequency) of counting pulses per DC541 cycle.

Example:

- Number of counting pulses (frequency) = 40 kHz = 40000 increments/s = 40 increments/ms
- Cycle time of the DC541 = 100 µs
- LIM_MIN = 0

-> Number of counting pulses per DC541 cycle: 40 increments/ms = 4 increments/100 µs
-> LIM_MAX > 8

Input SET can be used to set the counter to the value CNT_SET. This counting value is kept as long as input SET = TRUE.

If the reference point approach is enabled at input EN_REF, the counter is set to the value of input CNT_SET when a rising edge occurs on channel C2 or C3.

Using input EN_TOUCH, a touch trigger measurement is enabled. This means: With the rising edge on channel C2 or C3, the counting value is stored and displayed at output CNT_TOUCH. The validity of CNT_TOUCH is indicated by output RDY_TOUCH. This functionality can be used to determine the counter value with regard to an external event. The results are increment accurate.

Only one function may be enabled at a time, either the reference point approach or the touch trigger measurement. If both functions are enabled simultaneously or if the execution of one function is not yet completed when enabling the other function, a corresponding error message is displayed at the outputs ERR/ERNO.

To initiate a new reference point approach or touch trigger measurement, a positive edge at the corresponding enabling input is necessary.

If the zero track of an incremental transmitter is wired to channel C2 or C3, no touch trigger measurement may be performed in the region of the reference cam!

The device DC541 must be configured as counting device (counter mode).

The block DC541_32BIT_CNT has an integrated visualization visuDC541_32BIT_CNT which can be used to control all block functions in parallel to the user program, if input EN_VISU = TRUE. A detailed functional description of the visualization and how to integrate it can be found at the end of the block description.

The block **DC541_LIMIT** is used for limit value monitoring of the 32 bit counter. The block can be used to directly display various counting values of the 32 bit counter (DC541_32BIT_CNT) via binary outputs. Using the input SIGNAL you can determine whether the corresponding output is switched to FALSE or TRUE.

The time resolution of the block is < 100 µs, i.e., the result is increment accurate up to a frequency of 10 kHz.

The upper limit value LIM_MAX has to be higher than the lower limit value LIM_MIN. If the lower limit value LIM_MIN is higher than the upper limit value LIM_MAX, a corresponding error message is displayed at the outputs ERR/ERNO.

The device DC541 must be configured as counting device (counter mode) and channel C0 as 32 bit counter.

The inputs and outputs of the blocks DC541_32BIT_CNT and DC541_LIMIT are described in detail in the library documentation.

3.3.2 Configuration example: 32 bit up/down counter (encoder mode)

The configuration example described in this section is contained in the following folder on the Control Builder PS501 CD-ROM (V1.1 and later):

..\CD_AC500\Examples\DC541

File name:

DC541_DokuCounter_32BITEncoderExample_PM591_V11.pro

The 32 bit up/down counter in the operating mode "encoder mode" or "endless mode" corresponds to mode 7 (1 UpDown directional discriminator) of the high-speed counter of the digital input/output modules. However, the 32 bit counter of the DC541 additionally has one input for the zero track (reference point approach) and one touch trigger input.

In the configuration example, the counting pulses are therefore applied in parallel to the DC541 inputs and the counting inputs of the DC532.

Hardware configuration:

The example control system shall have the following configuration:

- Terminal base TB521 (two coupler slots)
- DC541 in coupler slot 1 (first slot on the left of the CPU)
- PM591-ETH CPU with internal Ethernet coupler (configuration using SYCON.net)
- I/O module DC532 on the I/O bus

Wiring:

The channels are connected as follows:

DC532 / C16 -------------- DC541 / C0
DC532 / C17 -------------- DC541 / C1
DC532 / C18 -------------- DC541 / C2
DC532 / C19 -------------- DC541 / C3
DC532 / C20 -------------- DC541 / C4
DC532 / C21 -------------- DC541 / C5
DC532 / C22 -------------- DC541 / C6
DC532 / C23 -------------- DC541 / C7
DC532 / C16 -------------- DC532 / C24
DC532 / C17 -------------- DC532 / C25

PLC configuration:

- DC541 in slot 1, operating mode "counter mode"

- Configuration:	- Channel	C0	32 bit counter
	- Channels	C1...C4	Input
	- Channel	C5	Output
	- Channels	C6...C7	Limit channel 0

- Specification of the Ethernet coupler as internal coupler (if available)
(Ethernet coupler configuration using SYCON.net)

- DC532 on the I/O bus / parameter "Fast counter" = 7-1 UpDown directional discriminator

Task configuration:

- Task 1: Cyclic program / Prio=10 / Interval = t#100ms / PLC_PRG
- Task 2: Simulation / Prio=15 / Interval = t#5ms / Simulation_Task

Purpose of the cyclic program PLC_PRG:

The cyclic program PLC_PRG contains the following functions:

- Reading the cycle of PLC_PRG
Calling of block TASK_INFO;
- Reading the configuration of the DC541
Calling of block DC541_GET_CFG
- Reading the status of the DC541
Calling of block DC541_STATE
- Reading/writing the static channels of the DC541
Calling of block DC541_IO
- Calling of the sequence control for the counters
Calling of program pro32BitCounter

The blocks DC541_GET_CFG, DC541_STATE and DC541_IO are contained in the library DC541_AC500_V11.lib and described in detail in the library documentation.

The block TASK_INFO is contained in the library SysInt_AC500_V1.0 and described in detail in the corresponding documentation.

The actual execution of the 32 bit up/down counter functionality is implemented in the program pro32BitCounter.

Purpose of the program pro32BitCounter:

The program pro32BitCounter executes the following step chain:

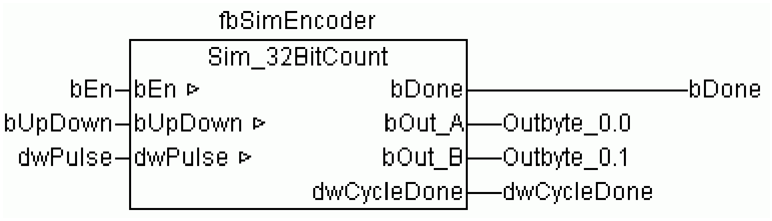
Step (byStep)	DC541_32BIT_CNT	DC532 / CNT_IO
0	Initialization, set counter to 250	Initialization, set counter to 250
1	1000 pulses / endless mode / UP	1000 pulses / UP
2	Wait 5 seconds	Wait 5 seconds
3	1000 pulses / endless mode / DOWN	1000 pulses / DOWN
4	Wait 5 seconds	Wait 5 seconds
5	3000 pulses / limit mode between 250 and 1000 (4 executions)	3000 pulses / UP
6	Wait 5 seconds	Wait 5 seconds
7	3000 pulses / limit mode between 250 and 1000 (4 executions)	3000 pulses / DOWN
8	Wait 5 seconds	Wait 5 seconds
9	Endless mode / UP Start reference point approach	No significance
10	Set reference input (C2 = TRUE) and stop pulse output	No significance
11	Wait 5 seconds	No significance
12	Terminate reference point approach (EN_REF = FALSE) Reset reference input (C2 = FALSE)	No significance
13	Wait 5 seconds	No significance
14	Enable touch trigger (EN_TOUCH) and start pulse output	No significance
15	Set touch trigger input (C3 = TRUE)	No significance
16	Wait 5 seconds	No significance
17	Terminate touch trigger (EN_TOUCH = FALSE) Reset touch trigger input (C3 = FALSE)	No significance
200	Manual operation	Manual operation
249	Program end, restart from step 0 after 5 seconds	Program end, restart from step 0 after 5 seconds

The wait steps (5 seconds) have been inserted to allow an easier observation of the execution.

The channels C6 and C7 of the DC541 are configured as "Limit channel 0". Depending on the settings for LIM_MIN (= 500), LIM_MAX (= 750) and SIGNAL, the following values result for the channels C6 and C7:

Channel	DC541_32BITCNT.ACT_CNT		
	00...499	500...750	751...
C6 (SIGNAL = TRUE)	FALSE	TRUE	FALSE
C7 (SIGNAL = FALSE)	TRUE	FALSE	TRUE

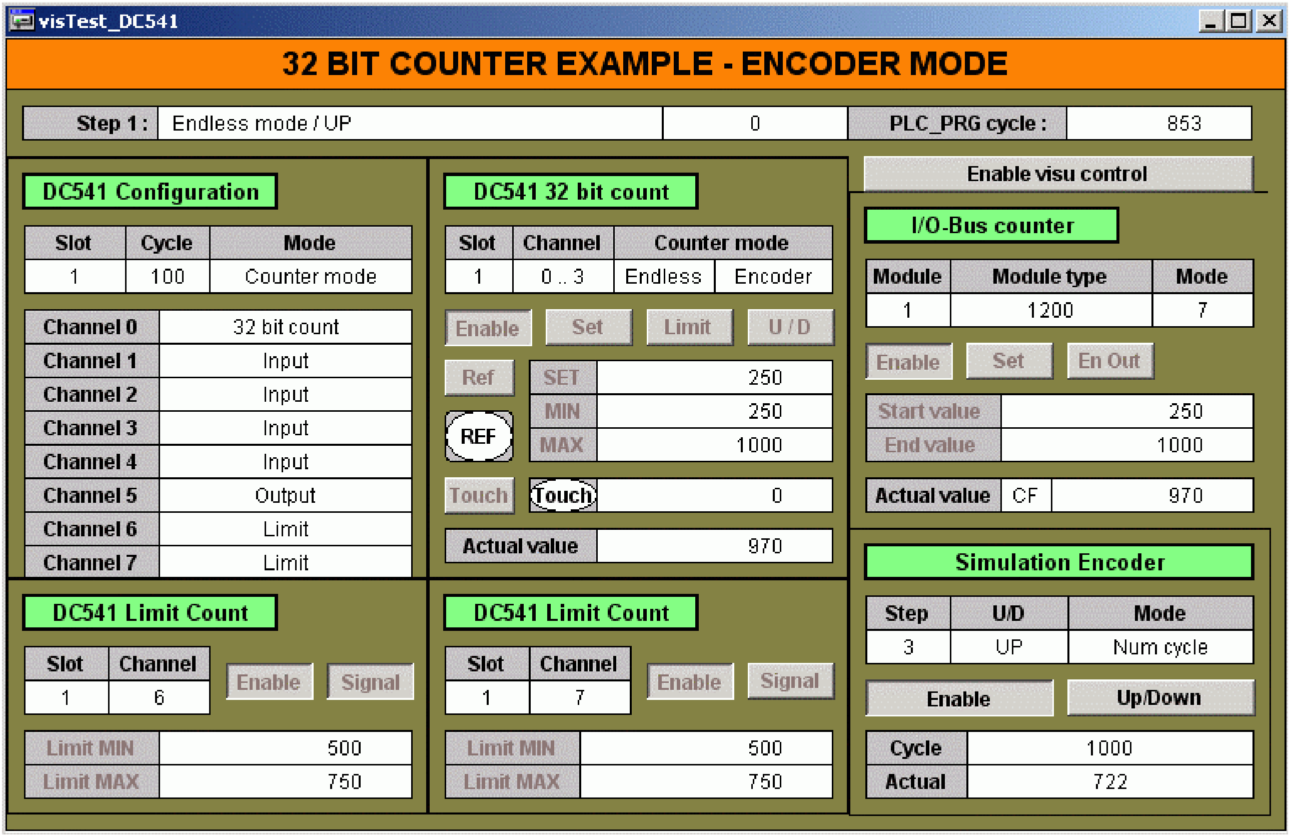
The block Sim_32BitCount is used to generate an adjustable number of pulses. Its representation in the function block diagram (FBD) is as follows:



Instance		fbSimuPulse	Instance name
bEn	Input/Output	BOOL	Enabling of the pulse output
bUpDown	Input/Output	BOOL	Selection UP / DOWN
dwPulse	Input/Output	DWORD	Number of pulses to be output
bDone	Output	BOOL	Completion message after the number of pulses specified at dwPulse or after every pulse if dwPulse = 0
bOut_A	Output	BOOL	Output track A
bOut_B	Output	BOOL	Output track B
dwCycleDone	Output	DWORD	Number of pulses output

In the example, the block Sim_32BitCount is called in a 5 ms task in order to generate the pulses with an offset of 90 degrees. The pulse output is enabled or stopped via input bEn. If input dwPulse = 0, the pulses are output continuously. If dwPulse > 0, only the specified number of pulses is output. When the specified number of pulses is reached, output bDone is set to TRUE.

The example program has a visualization implemented that displays all states:



32 BIT COUNTER EXAMPLE - ENCODER MODE

Step 1 : Endless mode / UP 0 PLC_PRG cycle : 853

DC541 Configuration

Slot	Cycle	Mode
1	100	Counter mode

Channel 0	32 bit count
Channel 1	Input
Channel 2	Input
Channel 3	Input
Channel 4	Input
Channel 5	Output
Channel 6	Limit
Channel 7	Limit

DC541 32 bit count

Slot	Channel	Counter mode	
1	0 .. 3	Endless	Encoder

SET	250
MIN	250
MAX	1000
Touch	0
Actual value	970

I/O-Bus counter

Module	Module type	Mode
1	1200	7

Start value	250
End value	1000
Actual value CF	970

DC541 Limit Count

Slot	Channel
1	6

Limit MIN	500
Limit MAX	750

DC541 Limit Count

Slot	Channel
1	7

Limit MIN	500
Limit MAX	750

Simulation Encoder

Step	U/D	Mode
3	UP	Num cycle

Cycle	1000
Actual	722

Clicking on the button <Enable visu control> (bEnVisuControl = TRUE) causes the program to jump from the current step to step 200 (manual operation). Then, the operation of the blocks is done via the corresponding pushbuttons/switches of the individual blocks. When manual operation is switched off again (bEnVisuControl = FALSE), the program jumps to step 249 and restarts from step 0 after the wait time.

3.3.3 Configuration example: 32 bit up/down counter (up/down mode)

The configuration example described in this section is contained in the following folder on the Control Builder PS501 CD-ROM (V1.1 and later):

..\CD_AC500\Examples\DC541

File name:

DC541_DokuCounter_32BITUpDownExample_PM591_V11.pro

The configuration example for the "up/down" counting mode of the 32 bit up/down counter essentially corresponds to the example for the counting mode "encoder mode". Depending on the counting mode, the simulation task counts pulses alternately on C0 and C1.

Hardware configuration:

The example control system shall have the following configuration:

- Terminal base TB521 (two coupler slots)
- DC541 in coupler slot 1 (first slot on the left of the CPU)
- PM591-ETH CPU with internal Ethernet coupler (configuration using SYCON.net)
- I/O module DC532 on the I/O bus

Wiring:

The channels are connected as follows:

DC532 / C16 -------------- DC541 / C0
DC532 / C17 -------------- DC541 / C1
DC532 / C18 -------------- DC541 / C2
DC532 / C19 -------------- DC541 / C3
DC532 / C20 -------------- DC541 / C4
DC532 / C21 -------------- DC541 / C5
DC532 / C22 -------------- DC541 / C6
DC532 / C23 -------------- DC541 / C7
DC532 / C16 -------------- DC532 / C24
DC532 / C17 -------------- DC532 / C25

PLC configuration:

- DC541 in slot 1, operating mode "counter mode"

- Configuration:	- Channel	C0	32 bit counter
	- Channels	C1...C4	Input
	- Channel	C5	Output
	- Channels	C6...C7	Limit channel 0

- Specification of the Ethernet coupler as internal coupler (if available)
(Ethernet coupler configuration using SYCON.net)
- DC532 on the I/O bus / parameter "Fast counter" = 3-2 UpDown counters

Task configuration:

- Task 1: Cyclic program / Prio=10 / Interval = t#100ms / PLC_PRG
- Task 2: Simulation / Prio=15 / Interval = t#5ms / Simulation_Task

Purpose of the cyclic program PLC_PRG:

The cyclic program PLC_PRG contains the following functions:

- Reading the cycle of PLC_PRG
Calling of block TASK_INFO;
- Reading the configuration of the DC541
Calling of block DC541_GET_CFG
- Reading the status of the DC541
Calling of block DC541_STATE
- Reading/writing the static channels of the DC541
Calling of block DC541_IO
- Calling of the sequence control for the counters
Calling of program pro32BitCounter

The blocks DC541_GET_CFG, DC541_STATE and DC541_IO are contained in the library DC541_AC500_V11.lib and described in detail in the library documentation.

The block TASK_INFO is contained in the library SysInt_AC500_V1.0 and described in detail in the corresponding documentation.

The actual execution of the 32 bit up/down counter functionality is implemented in the program pro32BitCounter.

Purpose of the program pro32BitCounter:

The program pro32BitCounter executes the following step chain:

Step byStep	DC541_32BIT_CNT		DC532 / CNT_IO		
	Sequence	Act. value	Sequence	Actual value	
				1	2
0	Initialization, set counter to 0	0	Initialization, set counter Counter 1: UD1 = TRUE -> Down (backward) SET1 = 10000 Counter 2: UD2 = FALSE -> Up (forward) SET2 = 0	10000	0
1	1000 pulses / endless mode / UP Pulses on channel C1	1000	1000 pulses / UP Pulses on counter 2	10000	1000
2	Wait 5 seconds	1000	Wait 5 seconds	10000	1000
3	1000 pulses / endless mode / DOWN Pulses on channel C0	0	1000 pulses / DOWN Pulses on counter 1	9000	1000
4	Wait 5 seconds	300	Wait 5 seconds	9000	1000
5	Set value = 300 3000 pulses / UP / limit mode between 300 and 800 (6 executions)	800	3000 pulses / UP Pulses on counter 2	9000	4000
6	Wait 5 seconds	800	Wait 5 seconds	9000	4000
7	3000 pulses / DOWN / limit mode between 300 and 800 (6 executions)	300	3000 pulses / DOWN Pulses on counter 1	6000	4000
8	Wait 5 seconds	300	Wait 5 seconds	6000	4000
9	Endless mode / UP Start reference point approach	810	No significance, counter 2 continues counting	6000	4510
10	Set reference input (C2 = TRUE) and stop pulse output Counter is set to SET = 300	300	No significance	6000	4510

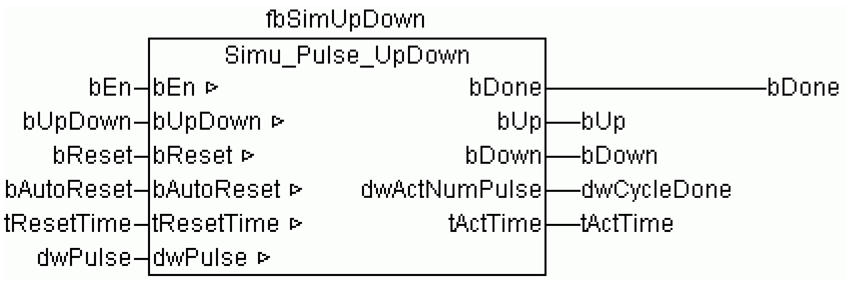
11	Wait 5 seconds	300	No significance	6000	4510
12	Terminate reference point approach (EN_REF = FALSE) Reset reference input (C2 = FALSE)	300	No significance	6000	4510
13	Wait 5 seconds	300	No significance	6000	4510
14	Enable touch trigger (EN_TOUCH) and start pulse output	810	No significance, counter 2 continues counting	6000	5320
15	Set touch trigger input (C3 = TRUE) In the example, touch trigger output CNT_TOUCH = 810	810	No significance	6000	5320
16	Wait 5 seconds	1350	No significance	6000	5560
17	Terminate touch trigger (EN_TOUCH = FALSE) Reset touch trigger input (C3 = FALSE)	1350	No significance	6000	5560
200	Manual operation	xxx	Manual operation	xxx	xxx
249	Program end, restart from step 0 after 5 seconds	1350	Program end, restart from step 0 after 5 seconds	6000	5560

The wait steps (5 seconds) have been inserted to allow an easier observation of the execution.

The channels C6 and C7 of the DC541 are configured as "Limit channel 0". Depending on the settings for LIM_MIN (= 500), LIM_MAX (= 750) and SIGNAL, the following values result for the channels C6 and C7:

Channel	**DC541_32BIT_CNT.ACT_CNT**		
	0...499	**500...750**	**751...**
C6 (SIGNAL = TRUE)	FALSE	TRUE	FALSE
C7 (SIGNAL = FALSE)	TRUE	FALSE	TRUE

The block Sim_32BitCount is used to generate an adjustable number of pulses. Its representation in the function block diagram (FBD) is as follows:

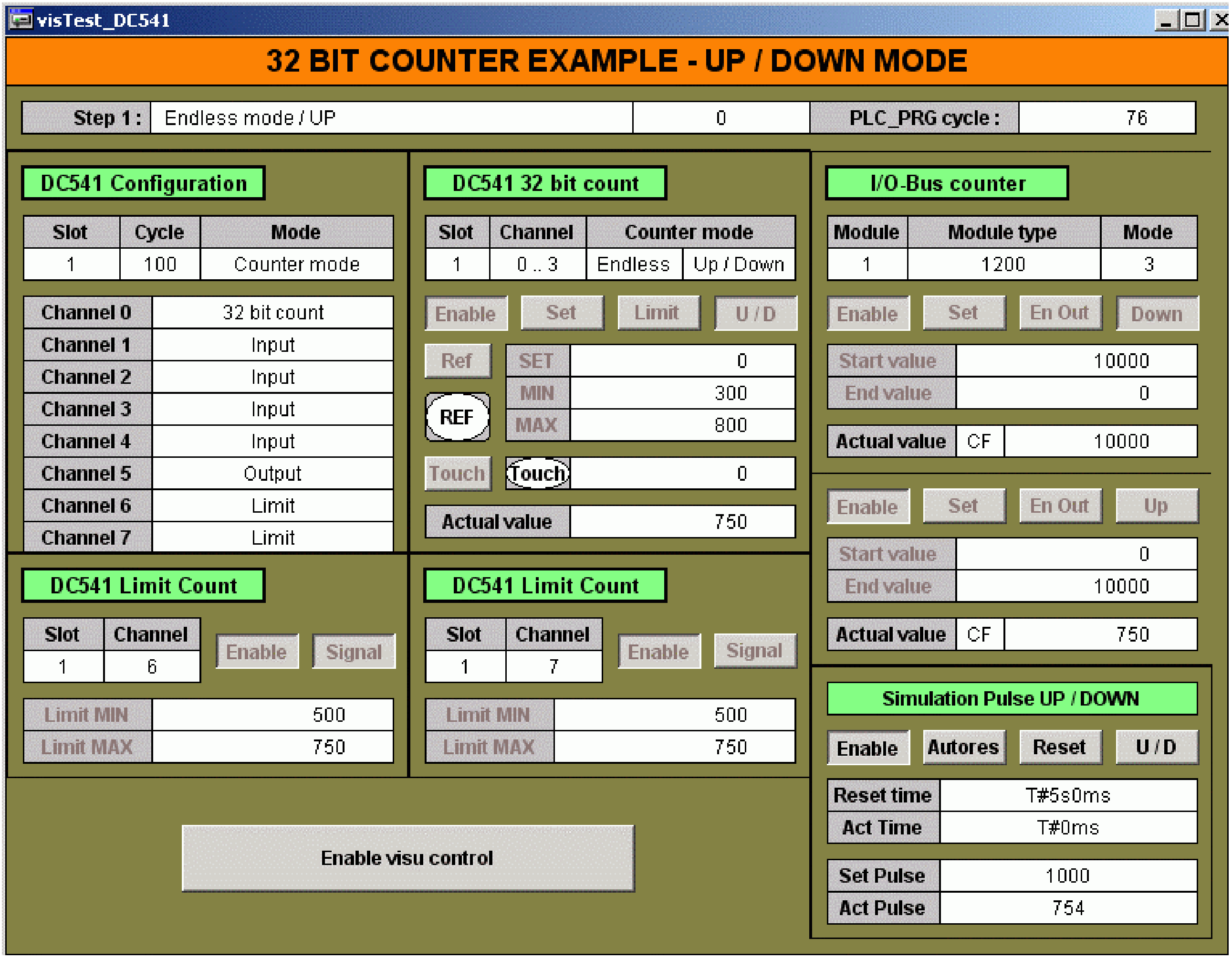


Instance		**fbSimuPulse**	**Instance name**
BEn	Input/Output	BOOL	Enabling of the pulse output
bUpDown	Input/Output	BOOL	Selection UP / DOWN
bReset	Input/Output	BOOL	TRUE = Reset of the pulse counter, bDone = TRUE
bAutoReset	Input/Output	BOOL	TRUE and cyclic mode - The time tResetTime is started when the number of pulses set with dwPulse is reached. After this time, the pulse output is restarted again.
tResetTime	Input/Output	TIME	Wait time until restart, if bAutoReset = TRUE

dwPulse	Input/Output	DWORD	Number of pulses to be output: = 0: Endless mode (pulse output continues until bEn = FALSE or bReset = TRUE) > 0: Cyclic mode (output of the specified number of pulses)
bDone	Output	BOOL	Completion message after the number of pulses specified at dwPulse or after bReset if dwPulse = 0
BUp	Output	BOOL	Output UP (forward)
bDown	Output	BOOL	Output DOWN (backward)
dwActNumPulse	Output	DWORD	Number of pulses output
tActTime	Output	TIME	Elapsed time in [ms] while tResetTime is running

In the example, the block Sim_32BitCount is called in a 5 ms task. The pulses are applied at output bUp or bDown depending on the value present at input bUpDown. The pulse output is enabled or stopped via input bEn. If input dwPulse = 0, the output of pulses is performed continuously. If dwPulse > 0, only the specified number of pulses is output. When the specified number of pulses is reached, output bDone is set to TRUE.

The example program has a visualization implemented that displays all states:



visTest_DC541

32 BIT COUNTER EXAMPLE - UP / DOWN MODE

Step 1 : Endless mode / UP 0 PLC_PRG cycle : 76

DC541 Configuration

Slot	Cycle	Mode
1	100	Counter mode

Channel 0	32 bit count
Channel 1	Input
Channel 2	Input
Channel 3	Input
Channel 4	Input
Channel 5	Output
Channel 6	Limit
Channel 7	Limit

DC541 32 bit count

Slot	Channel	Counter mode	
1	0 .. 3	Endless	Up / Down

Enable Set Limit U / D

Ref

SET	0
MIN	300
MAX	800

REF

Touch Touch 0

Actual value 750

I/O-Bus counter

Module	Module type	Mode
1	1200	3

Enable Set En Out Down

Start value	10000
End value	0

Actual value CF 10000

Enable Set En Out Up

Start value	0
End value	10000

Actual value CF 750

DC541 Limit Count

Slot	Channel
1	6

Enable Signal

Limit MIN	500
Limit MAX	750

DC541 Limit Count

Slot	Channel
1	7

Enable Signal

Limit MIN	500
Limit MAX	750

Simulation Pulse UP / DOWN

Enable Autores Reset U / D

Reset time	T#5s0ms
Act Time	T#0ms

Set Pulse	1000
Act Pulse	754

Enable visu control

Clicking on the button <Enable visu control> (bEnVisuControl = TRUE) causes the program to jump from the current step to step 200 (manual operation). Then, the operation of the blocks is done via the corresponding pushbuttons/switches of the individual blocks. When manual operation is switched off again (bEnVisuControl = FALSE), the program jumps to step 249 and restarts from step 0 after the wait time.

3.4 The 32 bit forward counter of module DC541-CM

3.4.1 Description of the module's forward counter functionality

The 32 bit forward counter functionality is provided by the block DC541_FWD_CNT.

The block DC541_FWD_CNT provides a 32 bit forward counter which is able to count a maximum frequency of 50 kHz at the inputs C0 and C1 or 5 kHz at the inputs C2...C7. In the DC541, the counter is implemented as a 16 bit counter. The actual counter value ACT_CNT is calculated by the block by adding the counter differences that occur within the individual cycles. In order not to loose any counting pulses, the block has to be called cyclically with at least the following interval:

- Channel 0...1: 50 kHz max. -> 32767 / 50 = 655 ms
- Channel 2...7: 5 kHz max. -> 32767 / 5 = 6550 ms

Using the counter e.g. in a 100 ms task will prevent any loss of counting pulses.

The counter can be used in two operating modes:

- Infinite counter (endless mode)
- Limiting counter (limit mode)

The operating mode is selected at input EN_LIM.

If EN_LIM = FALSE, the counter operates as infinite counter (endless mode). An overflow occurs corresponding to the 32 bit value at 16#FFFFFFFF = 4 294 967 295. In this mode, any exceeding of the limit value LIM_MAX or falling below the limit value LIM_MIN is displayed at the outputs MAX_LIM or MIN_LIM.

If EN_LIM = TRUE (limit mode), the counting range is between the limit values LIM_MIN and LIM_MAX. In case of an overflow, i.e. if LIM_MAX is reached, the counter restarts again at LIM_MIN.

The upper limit value LIM_MAX has to be higher than the lower limit value LIM_MIN. If the lower limit value LIM_MIN is higher than the upper limit value LIM_MAX, a corresponding error message is applied at the outputs ERR/ERNO.

The device DC541 must be configured as counting device (counter mode).

The block DC541_FWD_CNT has an integrated visualization visuDC541_FWD_CNT which can be used to control all block functions in parallel to the user program, if input EN_VISU = TRUE.

The inputs and outputs of the block DC541_FWD_CNT are described in detail in the library documentation.

3.4.2 Configuration example: 32 bit forward counter

The configuration example described in this section is contained in the following folder on the Control Builder PS501 CD-ROM (V1.1 and later):

..\CD_AC500\Examples\DC541

File name:

DC541_DokuCounter_32BitForwardExample_PM591_V11.pro

All of the 8 channels of the DC541 can be used as count-up counter. In the configuration example, all 8 channels of the DC541 are configured as 32 bit forward counter (count-up). The channels C0...C3 operate as infinite counters (endless mode), the channels C4...C7 as limit counters (limit mode).

The 32 bit count-up counter configured as infinite counter (endless mode) corresponds to mode 1 (1 Up counter) of the high-speed counter of the digital input/output modules. In the configuration example, the counting pulses for the first forward counter are therefore applied in parallel to input C0 of the DC541 and counting input C24 of the DC532.

Hardware configuration:

The example control system shall have the following configuration:

- Terminal base TB521 (two coupler slots)
- DC541 in coupler slot 1 (first slot on the left of the CPU)
- PM591-ETH CPU with internal Ethernet coupler (configuration using SYCON.net)
- I/O module DC532 on the I/O bus

Wiring:

The channels are connected as follows:

```
DC532 / C16 -------------- DC541 / C0
DC532 / C17 -------------- DC541 / C1
DC532 / C18 -------------- DC541 / C2
DC532 / C19 -------------- DC541 / C3
DC532 / C20 -------------- DC541 / C4
DC532 / C21 -------------- DC541 / C5
DC532 / C22 -------------- DC541 / C6
DC532 / C23 -------------- DC541 / C7
DC532 / C16 -------------- DC532 / C24
```

PLC configuration:

- DC541 in slot 1, operating mode "Counter mode"
- Configuration: - Channel C0..C7 Forward counter
- Specification of the Ethernet coupler as internal coupler (if available)
(Ethernet coupler configuration using SYCON.net)
- DC532 on the I/O bus / parameter "Fast counter" = 1-1 Up counter

Task configuration:

- Task 1: Cyclic program / Prio = 10 / Interval = t#100ms / PLC_PRG
- Task 2: Simulation / Prio = 15 / Interval = t#5ms / Simulation_Task

Purpose of the cyclic program PLC_PRG:

The cyclic program PLC_PRG contains the following functions:

- Reading the cycle of PLC_PRG
Calling of block TASK_INFO;
- Reading the configuration of the DC541
Calling of block DC541_GET_CFG
- Reading the status of the DC541
Calling of block DC541_STATE
- Reading/writing the static channels of the DC541
Calling of block DC541_IO
- Calling of the sequence control for the counters
Calling of program proForwardCounter

The blocks DC541_GET_CFG, DC541_STATE and DC541_IO are contained in the library DC541_AC500_V11.lib and described in detail in the library documentation.

The block TASK_INFO is contained in the library SysInt_AC500_V1.0 and described in detail in the corresponding documentation.

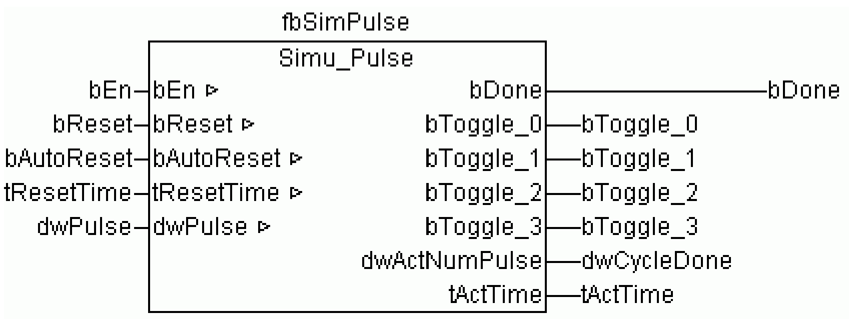
The actual execution of the 32 bit forward counter functionality is implemented in the program proForwardCounter.

Purpose of the program proForwardCounter:

The program proForwardCounter executes the following step chain:

Counter block	DC541_FWD_CNT CNT_IO								CNT_IO
Step|Chan	**C0**	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**	**1**
0 | Action	Init: SET = 0, endless counter, limit values MIN = 300 / MAX = 1300				Init: SET = 0, limit counter, limit values MIN = 300 / MAX = 1300				Init
| Value	0	0	0	0	0	0	0	0	0
1 | Action	Reset of SET input								
| Value	0	0	0	0	300	300	300	300	0
2 | Action	Start of pulse output - 2000 pulses								
| Value	0	0	0	0	300	300	300	300	0
3 | Action	Wait until pulse output is completed								
| Value	2000	1000	500	250	1299	1300	800	550	2000
4 | Action	Selection last step: byStep = 249								
| Value	2000	1000	500	250	1299	1300	800	550	2000
200|Action	Manual operation								
| Value	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx
249|Action	Wait time 5 seconds, then restart from step 0								
| Value	2000	1000	500	250	1299	1300	800	550	2000

The block Simu_Pulse is used to generate an adjustable number of pulses. Its representation in the function block diagram (FBD) is as follows:



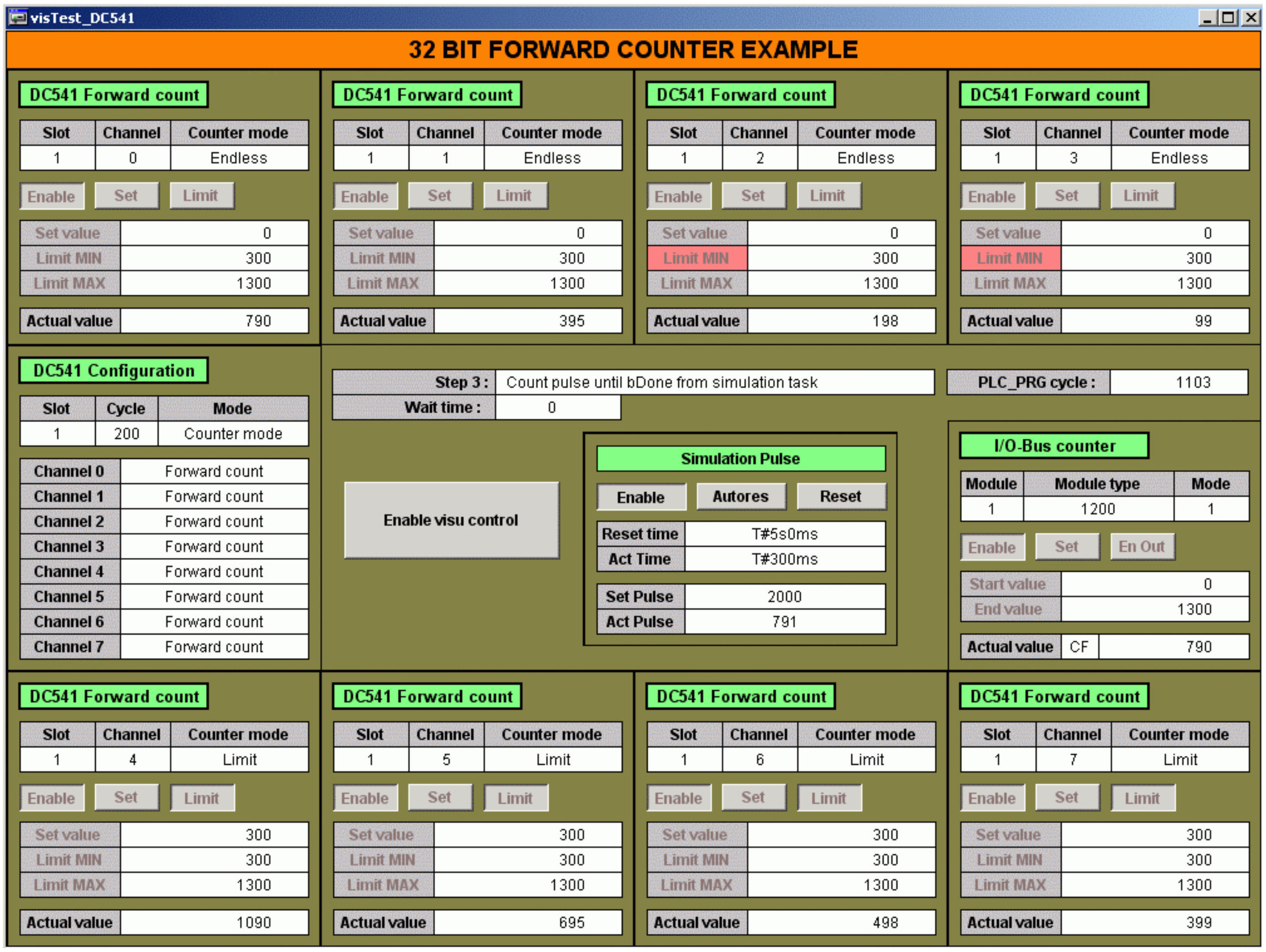
Instance		fbSimuPulse	Instance name
Ben	Input/Output	BOOL	Enabling of the pulse output
bReset	Input/Output	BOOL	TRUE = Reset of the pulse counter, bDone = TRUE
bAutoReset	Input/Output	BOOL	TRUE and cyclic mode - The time tResetTime is started when the number of pulses set with dwPulse is reached. After this time, the pulse output is restarted again.
tResetTime	Input/Output	TIME	Wait time until restart, if bAutoReset = TRUE
dwPulse	Input/Output	DWORD	Number of pulses to be output: = 0: Endless mode (pulse output continues until bEn = FALSE or bReset = TRUE) > 0: Cyclic mode (output of the specified number of pulses)
Bdone	Output	BOOL	Completion message after the number of pulses specified at dwPulse or after bReset if dwPulse = 0
bToggle_0	Output	BOOL	Output: Edge with each clock cycle

bToggle_1	Output	BOOL	Output: Edge with each 2. clock cycle
bToggle_2	Output	BOOL	Output: Edge with each 4. clock cycle
bToggle_3	Output	BOOL	Output: Edge with each 8. clock cycle
dwActNumPulse	Output	DWORD	Number of pulses output
tActTime	Output	TIME	Elapsed time in [ms] while tResetTime is running

In the example, the block Simu_Pulse is called in a 5 ms task. The pulse output is enabled or stopped via input bEn. If input dwPulse = 0, the output of pulses is performed continuously. If dwPulse > 0, only the specified number of pulses is output. When the specified number of pulses is reached, output bDone is set to TRUE.

In the example, the block is called with dwPulse = 2000. The wait time function is not used.

The example program has a visualization implemented that displays all states:



visTest_DC541

32 BIT FORWARD COUNTER EXAMPLE

DC541 Forward count

Slot	Channel	Counter mode	Set value	Limit MIN	Limit MAX	Actual value
1	0	Endless	0	300	1300	790
1	1	Endless	0	300	1300	395
1	2	Endless	0	300	1300	198
1	3	Endless	0	300	1300	99
1	4	Limit	300	300	1300	1090
1	5	Limit	300	300	1300	695
1	6	Limit	300	300	1300	498
1	7	Limit	300	300	1300	399

Enable Set Limit

DC541 Configuration

Slot	Cycle	Mode
1	200	Counter mode

Channel 0	Forward count
Channel 1	Forward count
Channel 2	Forward count
Channel 3	Forward count
Channel 4	Forward count
Channel 5	Forward count
Channel 6	Forward count
Channel 7	Forward count

Step 3 :	Count pulse until bDone from simulation task
Wait time :	0

PLC_PRG cycle : 1103

Enable visu control

Simulation Pulse

Enable Autores Reset

Reset time	T#5s0ms
Act Time	T#300ms
Set Pulse	2000
Act Pulse	791

I/O-Bus counter

Module	Module type	Mode
1	1200	1

Enable Set En Out

Start value		0
End value		1300
Actual value	CF	790

Clicking on the button <Enable visu control> (bEnVisuControl = TRUE) causes the program to jump from the current step to step 200 (manual operation). Then, the operation of the blocks is done via the corresponding pushbuttons/switches of the individual blocks. When manual operation is switched off again (bEnVisuControl = FALSE), the program jumps to step 249 and restarts from step 0 after the wait time.

3.5 Pulse width modulation (PWM) using the DC541-CM

3.5.1 Description of the module's PWM functionality

The pulse width modulation functionality of the DC541 is provided by the block **DC541_PWM**.

The block DC541_PWM outputs a pulsed signal with an adjustable on-off ratio. The adjustment of the on and off times is done as 8 bit numbers.

The minimum switching time is specified at input CYCLE, i.e. if an output has been switched to FALSE or TRUE by the PWM, this output remains in this state for at least this time (CYCLE µs).

The minimum time specified at input CYCLE must not be shorter than the cycle time of the device DC541. Depending on its configuration, the cycle time of the DC541 can be 50, 100 or 200 µs. The cycle time can be polled using the block DC541_GET_CFG (output CYCLE).

Examples:

PULSE	PAUSE	CYCLE	Result (x = number of cycles of the DC541)
Cycle time of DC541 = 50 µs			
1	2	500	10 x TRUE / 20 x FALSE / 10 x TRUE / 20 x FALSE / … i.e. 500 µs = TRUE and 1000 µs = FALSE
4	8	500	10 x TRUE / 20 x FALSE / 10 x TRUE / 20 x FALSE / … i.e. 500 µs = TRUE and 1000 µs = FALSE (as in example 1, i.e. ratio 1 : 2)
3	2	3000	90 x TRUE / 60 x FALSE / 90 x TRUE / 60 x FALSE / … i.e. 4500 µs = TRUE and 3000 µs = FALSE
Cycle time of DC541 = 100 µs			
1	2	500	5 x TRUE / 10 x FALSE / 5 x TRUE / 10 x FALSE / … i.e. 500 µs = TRUE and 1000 µs = FALSE
4	8	500	5 x TRUE / 10 x FALSE / 5 x TRUE / 10 x FALSE / … i.e. 500 µs = TRUE and 1000 µs = FALSE (as in example 1, i.e. ratio 1 : 2)
3	2	30000	45 x TRUE / 30 x FALSE / 45 x TRUE / 30 x FALSE / … i.e. 4500 µs = TRUE and 3000 µs = FALSE
Cycle time of DC541 = 200 µs			
1	2	500	3 x TRUE / 6 x FALSE / 3 x TRUE / 6 x FALSE /... i.e. 600 µs = TRUE, 1200 µs = FALSE
4	8	500	3 x TRUE / 6 x FALSE / 3 x TRUE / 6 x FALSE / ... i.e. 600 µs = TRUE, 1200 µs = FALSE (as in example 1, i.e. ratio 1 : 2)
3	2	3000	22 x TRUE / 15 x FALSE / 23 x TRUE / 15 x FALSE / … i.e. first 4400 µs = TRUE and 3000 µs = FALSE and then 4600 µs = TRUE and 3000 µs = FALSE

The device DC541 must be configured as counting device (counter mode).

The inputs and outputs of the block DC541_PWM are described in detail in the documentation of the library DC541_AC500_V11.LIB.

3.5.2 Configuration example: Pulse width modulation (PWM)

The configuration example described in this section is contained in the following folder on the Control Builder PS501 CD-ROM (V1.1 and later):

..\CD_AC500\Examples\DC541

File name:

DC541_DokuCounter_PWM_FREQ_Example_PM591_V11.pro

In the configuration example, channel 0 of the DC541 is configured for pulse width modulation (PWM). The output signal is measured using the function "Time and frequency measurement" of the DC541. This function is described in the next section.

The following on-off ratio shall be used:

PULSE	**PAUSE**	**CYCLE**	**Result (x = number of cycles of the DC541)**
Cycle time of DC541 = 100 µs			
1	2	2000	20 x TRUE / 40 x FALSE / 20 x TRUE / 40 x FALSE / … i.e. 2000 µs = TRUE and 4000 µs = FALSE

Hardware configuration:

The example control system shall have the following configuration:

- Terminal base TB521 (two coupler slots)
- DC541 in coupler slot 1 (first slot on the left of the CPU)
- PM591-ETH CPU with internal Ethernet coupler (configuration using SYCON.net)
- I/O module DC532 on the I/O bus

Wiring:

The channels are connected as follows:

DC541 / C0 -------------- DC541 / C1

PLC configuration:

- DC541 in slot 1, operating mode "counter mode"

- Configuration:	- Channel	C0	PWM
		C1	FREQ
		C2...C7	Input

- Specification of the Ethernet coupler as internal coupler (if available)
(Ethernet coupler configuration using SYCON.net)

Task configuration:

- Task 1: Cyclic program / Prio = 10 / Interval = t#1ms / PLC_PRG

Purpose of the cyclic program PLC_PRG:

The cyclic program PLC_PRG contains the following functions:

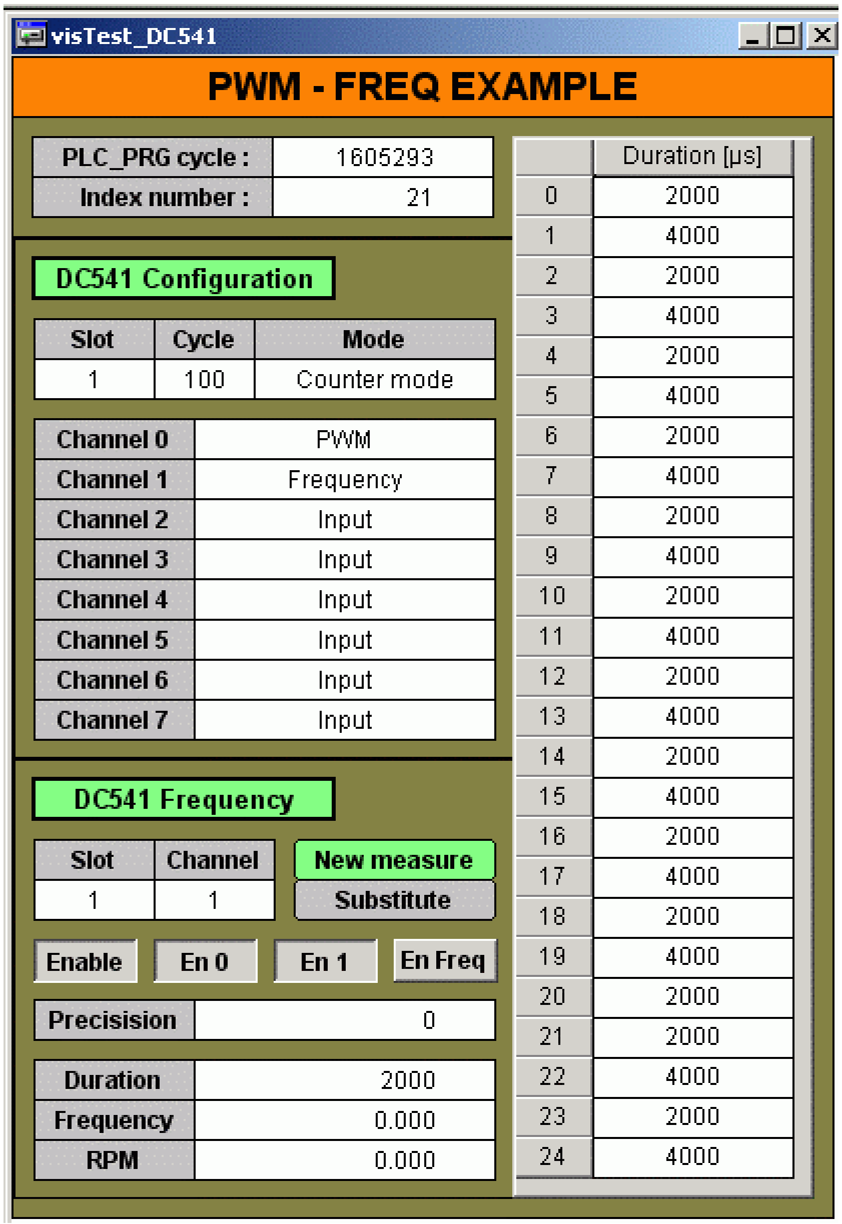
- Reading the cycle of PLC_PRG
Calling of block TASK_INFO;
- Reading the configuration of the DC541
Calling of block DC541_GET_CFG
- Reading the status of the DC541
Calling of block DC541_STATE
- Reading/writing the static channels of the DC541
Calling of block DC541_IO
- Calling of the sequence control for PWM and FREQ
Calling of program proPWM_FREQ

The blocks DC541_GET_CFG, DC541_STATE and DC541_IO are contained in the library DC541_AC500_V11.lib and described in detail in the library documentation.

The block TASK_INFO is contained in the library SysInt_AC500_V1.0 and described in detail in the corresponding documentation.

Calling the pulse width modulation functionality as well as measurement and acquisition of measured values are performed in the program proPWM_FREQ. The program proPWM_FREQ contains the calls for the function blocks DC541_PWM and DC541_FREQ as well as the acquisition of the measured values. The function block DC541_FREQ is configured in a way that it measures the time between each edge change.

The example program has a visualization implemented that displays all states:



PLC_PRG cycle :	1605293
Index number :	21

Slot	Cycle	Mode
1	100	Counter mode

Channel 0	PWM
Channel 1	Frequency
Channel 2	Input
Channel 3	Input
Channel 4	Input
Channel 5	Input
Channel 6	Input
Channel 7	Input

Slot	Channel
1	1

Precisision	0
Duration	2000
Frequency	0.000
RPM	0.000

	Duration [µs]
0	2000
1	4000
2	2000
3	4000
4	2000
5	4000
6	2000
7	4000
8	2000
9	4000
10	2000
11	4000
12	2000
13	4000
14	2000
15	4000
16	2000
17	4000
18	2000
19	4000
20	2000
21	2000
22	4000
23	2000
24	4000

Input EN_VISU of the function block DC541_FREQ is TRUE. Therefore, the inputs of the block can be modified using the buttons <Enable>, <En 0>, <En 1> and <En Freq> in the visualization.

The measured values are 2000, 4000 or 6000 µs depending on which edges were considered for measurement.

3.6 Time and frequency measurement using the DC541-CM

3.6.1 Description of the module's time and frequency measurement functionality

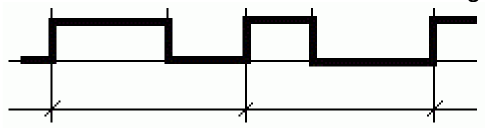
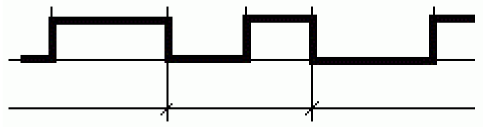
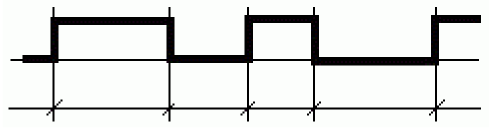
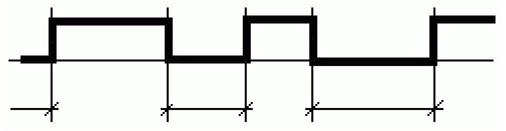
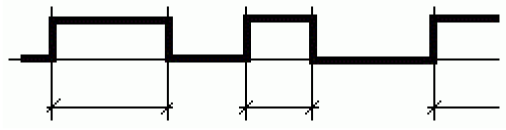
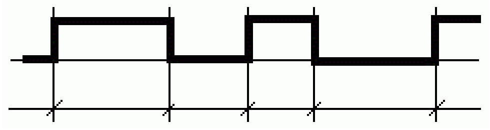
The time and frequency measurement functionality of the DC541 is provided by the block **DC541_FREQ**.

The block DC541_FREQ is used to measure times, frequencies and rotational speeds with a resolution of 100 µs.

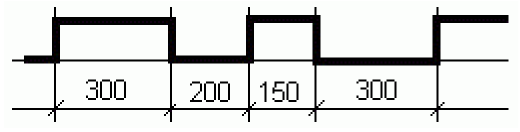
It is able to measure frequencies from 0 up to 2000 Hz (2 kHz). In order to obtain a precise measurement of frequencies > 50 Hz, a correspondingly high accuracy setting has to be chosen. It is recommended to use an accuracy of PREC = 1000, i.e. 0.001.

This block has to be called cyclically, one time per second at least.

The inputs EN_0, EN_1 and EN_FREQ are used to determine the edges to be measured. If input EN_FREQ = TRUE, the frequency and the rotational speed are calculated in addition to the time measurement.

EN_0	EN_1	EN_FREQ	Edges measured	FREQ/RPM
FALSE	FALSE	TRUE	No measurement is performed.	yes
FALSE	TRUE	TRUE	Measurement of time between two rising edges 	yes
TRUE	FALSE	TRUE	Measurement of time between two falling edges 	yes
TRUE	TRUE	TRUE	Measurement of time between any two edges 	yes
FALSE	FALSE	FALSE	No measurement is performed.	no
FALSE	TRUE	FALSE	Measurement of time between the falling edge and the subsequent rising edge 	no
TRUE	FALSE	FALSE	Measurement of time between the rising edge and the subsequent falling edge 	no
TRUE	TRUE	FALSE	Measurement of time between any two edges 	no

The following example shows the different time measurement results depending on the values applied to the inputs EN_0, EN_1 and EN_FREQ.



EN_0	EN_1	EN_FREQ	Time measurement (DUR) in [µs]			
			1	2	3	4
FALSE	FALSE	TRUE	0	0	0	0
FALSE	TRUE	TRUE	-	500	-	450
TRUE	FALSE	TRUE	-	-	350	-
TRUE	TRUE	TRUE	300	200	150	300
FALSE	FALSE	FALSE	0	0	0	0
FALSE	TRUE	FALSE	300	-	150	-
TRUE	FALSE	FALSE	-	200	-	300
TRUE	TRUE	FALSE	300	200	150	300

Output NEW indicates that new measurement results are available.

Note: If indicated by NEW, the PLC program always reads the last measurement value. The measurement values are not buffered in the DC541.

The device DC541 must be configured as counting device (counter mode). Channel CH must be configured for frequency measurement.

The block DC541_FREQ has an integrated visualization visuDC541_FREQ which can be used to control all block functions in parallel to the user program, if input EN_VISU = TRUE.

The inputs and outputs of the block DC541_FREQ are described in detail in the documentation of the library DC541_AC500_V11.LIB.

3.6.2 Configuration example: Frequency output

The configuration example described in this section is contained in the following folder on the Control Builder PS501 CD-ROM (V1.1 and later):

..\CD_AC500\Examples\DC541

File name:

DC541_DokuCounter_FREQ_Example_PM591_V11.pro

In the configuration example, channel 0 of the DC541 is configured for frequency output. The output signal is measured using the function "Time and frequency measurement" of the DC541. The function frequency output is described in the next section.

Hardware configuration:

The example control system shall have the following configuration:

- Terminal base TB521 (two coupler slots)
- DC541 in coupler slot 1 (first slot on the left of the CPU)
- PM591-ETH CPU with internal Ethernet coupler (configuration using SYCON.net)
- I/O module DC532 on the I/O bus

Wiring:

The channels are connected as follows:

DC541 / C0 -------------- DC541 / C1

PLC configuration:

- DC541 in slot 1, operating mode "counter mode"

- Configuration:	- Channel	C0	Frequency output
		C1	Frequency measurement
		C2...C7	Input

- Specification of the Ethernet coupler as internal coupler (if available)
(Ethernet coupler configuration using SYCON.net)

Task configuration:

- Task 1: Cyclic program / Prio = 10 / Interval = t#5ms / PLC_PRG

Purpose of the cyclic program PLC_PRG:

The cyclic program PLC_PRG contains the following functions:

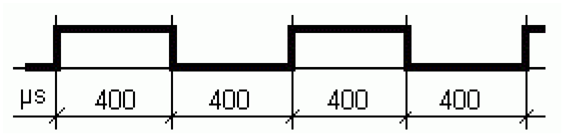
- Reading the cycle of PLC_PRG
Calling of block TASK_INFO;
- Reading the configuration of the DC541
Calling of block DC541_GET_CFG
- Reading the status of the DC541
Calling of block DC541_STATE
- Reading/writing the static channels of the DC541
Calling of block DC541_IO
- Calling of the sequence control for frequency output and measurement
Calling of program proFrequency

The blocks DC541_GET_CFG, DC541_STATE and DC541_IO are contained in the library DC541_AC500_V11.lib and described in detail in the library documentation.

The block TASK_INFO is contained in the library SysInt_AC500_V1.0 and described in detail in the corresponding documentation.

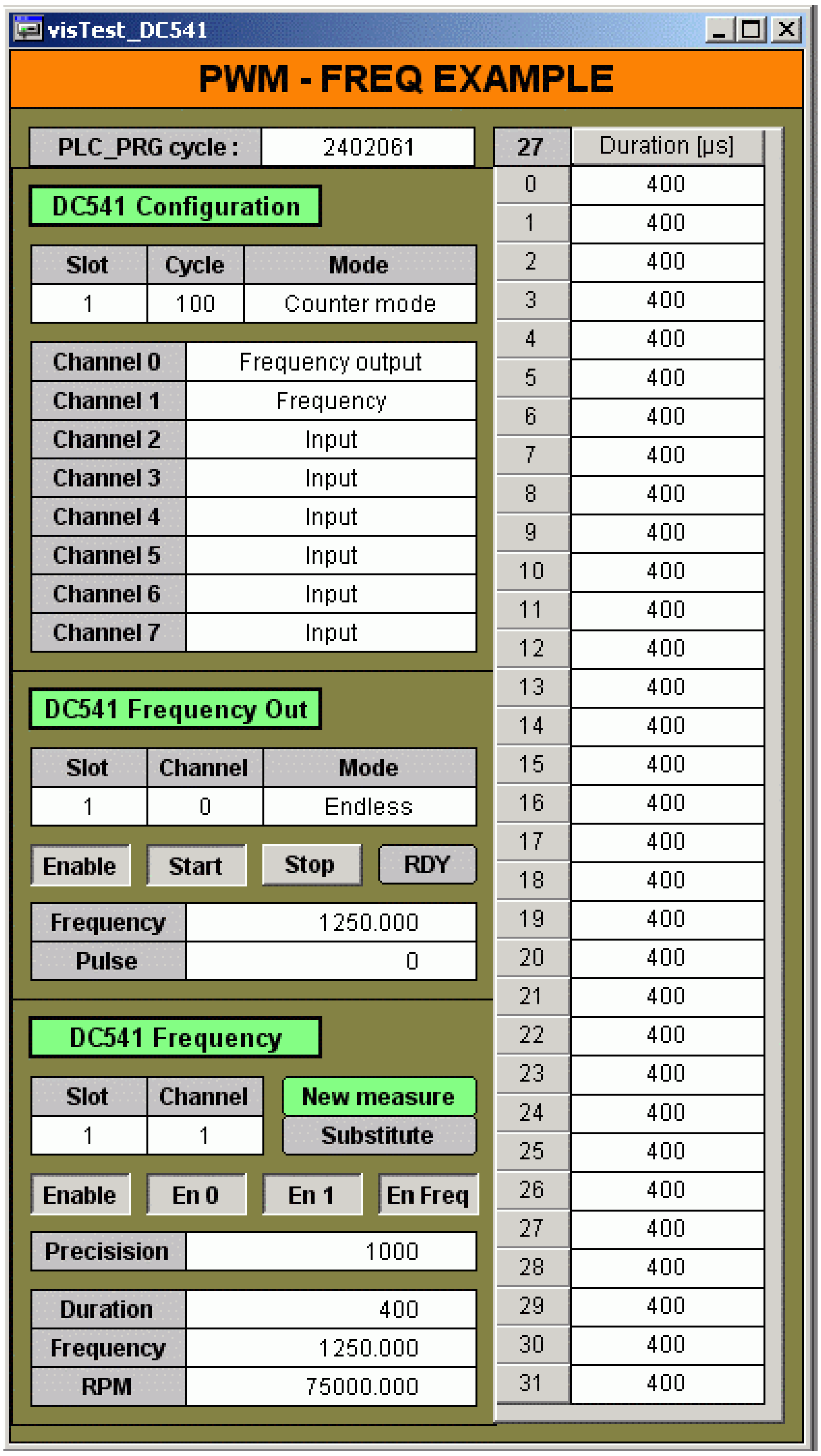
The calling of the frequency output functionality as well as the measurement and acquisition of measured values are performed in the program proFrequency. The program proFrequency contains the calls for the function blocks DC541_FREQ_OUT and DC541_FREQ as well as the acquisition of the measured values.

The following values are output if FREQ is set to 1250 (1250 Hz = 1.25 kHz):



The example program has a visualization implemented that displays all states:

Input EN_VISU of the function blocks DC541_FREQ_OUT and DC541_FREQ is TRUE. Therefore, the block inputs can be controlled using the buttons in the visualization.



visTest_DC541

PWM - FREQ EXAMPLE

PLC_PRG cycle :	2402061

DC541 Configuration

Slot	Cycle	Mode
1	100	Counter mode

Channel 0	Frequency output
Channel 1	Frequency
Channel 2	Input
Channel 3	Input
Channel 4	Input
Channel 5	Input
Channel 6	Input
Channel 7	Input

DC541 Frequency Out

Slot	Channel	Mode
1	0	Endless

Enable Start Stop RDY

Frequency	1250.000
Pulse	0

DC541 Frequency

Slot	Channel
1	1

New measure
Substitute

Enable En 0 En 1 En Freq

Precisision	1000

Duration	400
Frequency	1250.000
RPM	75000.000

27	Duration [µs]
0	400
1	400
2	400
3	400
4	400
5	400
6	400
7	400
8	400
9	400
10	400
11	400
12	400
13	400
14	400
15	400
16	400
17	400
18	400
19	400
20	400
21	400
22	400
23	400
24	400
25	400
26	400
27	400
28	400
29	400
30	400
31	400

3.7 Frequency output using the DC541-CM

3.7.1 Description of the module's frequency output functionality

The frequency output functionality of the DC541 is provided by the block **DC541_FREQ_OUT**.

The block DC541_FREQ_OUT is used to output pulses with a fixed frequency on one channel of the DC541. The module is able to output pulses with a frequency between 0.2 and 2.5 kHz. The pulse jitter depends on the cycle time of the DC541. The pulse length is always a multiple of the cycle time of the DC541.

In case of a presetting of PULSE = 0, the output of pulses is infinite. The pulse output is started with a positive edge at input START. The output is aborted if START = FALSE. A positive edge at input STOP interrupts the pulse output. The output is continued, if STOP = FALSE.

If input PULSE > 0, the block outputs the number of pulses specified at input PULSE with the frequency specified at input FREQ on the channel specified at input CH. After the block has output the number of pulses specified at PULSE, the output RDY becomes TRUE.

The device DC541 must be configured as counting device (counter mode). Channel CH must be configured for frequency output.

The block DC541_FREQ_OUT has an integrated visualization visuDC541_FREQ_OUT which can be used to control all block functions in parallel to the user program, if input EN_VISU = TRUE.

The inputs and outputs of the block DC541_FREQ_OUT are described in detail in the documentation of the library DC541_AC500_V11.LIB.

3.7.2 Configuration example: Frequency output

The configuration example described in this section is contained in the following folder on the Control Builder PS501 CD-ROM (V1.1 and later):

..\CD_AC500\Examples\DC541

File name:

DC541_DokuCounter_FREQ_Example_PM591_V11.pro

For frequency output, the same configuration example is used as for the time and frequency measurement. A description of the example can be found in the chapter about frequency and time measurement.

Link to chapter: 3.6.2 Configuration example: Frequency output

4 Index System Technology DC541-CM

System Description

AC500

Scalable PLC
for Individual Automation

System Technology
of the Ethernet Couplers

Ethernet

Contents "The Ethernet Coupler"

The Ethernet coupler

1 Ethernet and protocols

1.1 History

Ethernet is the most commonly used network technology for many fields of application. Based on the physics of Ethernet, numerous protocols are used today. One of the best-known representatives of these protocols is TCP/IP.

1.1.1 History of Ethernet

The history of today's Ethernet goes back to the early seventies. At that time, Ethernet was originally developed by Xerox Corporation to link a printer. Then, on 13 December 1979, Xerox Corporation had Ethernet patented as 'Multipoint Data Communication System with Collision Detection' in the US Register of Patents under the number 4 063 220.

As a result, the DIX group, a consortium consisting of DEC, Intel and Xerox, started its further development. The goal of this consortium was to establish the Ethernet technology as a LAN standard. For this purpose, the company-internal specification of the Ethernet LAN (which is today called Ethernet version 1) was published in September 1980.

Later, the American standardization Institute IEEE founded a study group named 802.3 to work out an internationally accepted standard on the basis of this publication. On 23 June 1983, Ethernet was approved for the first time as standard IEEE 802.3. In 1990, the 10BaseT standard IEEE 802.3i followed which specified the use of an unshielded twisted pair cable instead of the coaxial cable used before. Later, the original transfer rate of 10 Mbit/s was increased step by step and various other transfer media, e.g. optical fibres, were adapted with further standards.

1.1.2 History of TCP/IP protocols

The development of today's TCP/IP protocols (Transmission Control Protocol / Internet Protocol) originated in military and occurred independent of the development of the Ethernet. As early as the beginning sixties, the RAND corporation in USA accepted the challenge to develop a communication network which should be resistant against military attacks. Among other things, this concept included that the system had to deal without a susceptible central control. However, during the sixties this concept could not really make progress.

The actual history of TCP/IP protocols started in 1968, as a department of the American ministry of defense started a series of experiments for computer-to-computer connections based on multiplexed lines. In 1969, this resulted in the ARPANET project. The Advanced Research Projects Agency was the leading instance for the development of ARPAnet and at the same time provided the name. The ARPAnet enabled scientists to use computer data and programs on other people's computers located far away. In 1972, the ARPAnet already consisted of 37 nodes. The intensification of access control performed by the Pentagon resulted in a change of the name to DARPAnet (D from defense).

At that time, the net structures (e.g. telecommunication via satellite, local area networks (LAN)) were completely new and required the further development of the previous NCP protocol. Thereupon, the ARPA study group 'Internet Working Group' (INWG) established principles for the data transfer between independent networks, which produced the Protocol-for-Packet Network Intercommunication. This resulted in the development of the Kahn-Cerf protocols which became well-known under the name TCP/IP protocols.

On 1 January 1983, the previous NCP protocol was finally replaced by the set of TCP/IP protocols. Likewise in 1983, the DARPA net, which was previously strictly kept under the control of the Pentagon, was divided into a military and non-military network.

1.2 Ethernet

1.2.1 Frame formats

One fundamental part of the Ethernet specification is the arrangement of the data transfer format. When transferring data via Ethernet, the actual user data are preceded by a so-called preamble (which is among other things used to synchronize the receiver stations) as well as the hardware source and target address and a type length field. A checksum follows after the user data. All information mentioned above together constitute an Ethernet frame. During the development of the Ethernet, different types of frames arose. The following figure shows the structure of an Ethernet 802.3 frame.

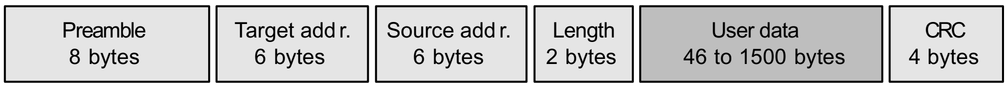


Figure 1-1: Structure of an Ethernet 802.3 frame

It has to be observed that the transferred user data do not inevitably contain only useful information. When transmitting data using a protocol above Ethernet (refer to 1.2.5), each protocol layer passed prior to the actual transmission supplements the original user data by its specific frame or header, so that the maximum number of actual user data is smaller, depending on the used protocols.

MAC address:

Each Ethernet terminal device that has the MAC layer functions implemented (refer to 1.2.5 Ethernet and TCP/IP) has a world-wide unique hardware and MAC address. In this 6 bytes address, the two most significant bits of the first byte have specific functions. The most significant bit is also called the I/G bit (Individual/Group bit) and indicates whether it is an individual world-wide unique address (unicast address, I/G bit = 0) or a group address (I/G bit = 1). The second most significant bit is called the G/L bit and indicates whether it is a globally or a locally administered MAC address. A GAA (Globally Administered Address, G/L bit = 0) is an address which is fixed programmed by the device manufacturer and has to be unique all over the world. An LAA (Locally Administered Address, G/L bit = 1) can be a MAC address which has been changed afterwards for the use within a network. For this, it has to be observed that a MAC address has to be unique within a network.

The first 3 bytes of a MAC address are the manufacturer-related address part. Using this value, the manufacturer of an Ethernet chip can be determined. Each manufacturer of Ethernet components has one or several pre-defined address ranges assigned he can use for his products. 3COM, for example, uses among others the MAC address range 02-60-8C-xx-xx-xx.

For Ethernet, the MAC address is represented in a canonical form. This representation starts with the least significant bit (LSB) and ends with the most significant bit (MSB) of a byte. The following figure shows a global unicast address of the manufacturer 3COM.

Canonical representation 02-60-8C-00-00-01
Binary representation 01000000-01100000-00110001-00000000-00000000-10000000

1.2.2 Bus access methods

Ethernet uses the CSMA/CD access method (Carrier Sense Multiple Access / Collision Detection). With this method, the station that wants to transmit data, first "listens" to the carrier whether data are currently being transmitted by another station (carrier sense). If the carrier is busy, the station later tries to access the carrier again. If the carrier is idle, the station starts the transmission.

With this method, particularly in greater networks, it can happen that several stations try to transmit at the same time (multiple access). As a result, they "listen" to the carrier, detect that the carrier is free and correspondingly start the transmission. This can cause collisions between the different data packages. This is why each station verifies whether a collision occurred during transmission (collision detection). If this is the case, the station aborts the transmission and then tries to send its data again after a wait time which is determined by a random generator.

Collisions within an Ethernet network do not cause loss of data, but they reduce the available bandwidth of the network. In practice, for a network with 30 stations on the bus, a net bandwidth of approx. 40 % is assumed. This means that a bandwidth of only approx. 4 Mbit/s is available in a network with a theoretical bandwidth of 10 Mbit/s, for instance. This has to be considered when planning an Ethernet network. The number of collisions can be reduced to a minimum if the network is carefully planned and if only suitable network components are used (refer to 1.4 Cabling and 1.5.4 Media converters).

1.2.3 Half duplex and full duplex

If communication is only possible in one direction (transmission or reception), this is called half duplex mode. However, the separate transmit and receive lines of today's twisted pair cabling for Ethernet networks also allow full duplex operation. In full duplex mode, the stations can simultaneously exchange data in both directions independent from each other. Due to this, the CSMA/CD method is not necessary in full duplex mode. Networks with more than two stations working in full duplex mode can only be implemented using switches because these switches establish peer-to-peer connections between the individual stations (refer to 1.4 Cabling).

1.2.4 Auto negotiation

Today, Ethernet uses transmission rates of 10, 100 or 1000 Mbit/s in half duplex or in full duplex mode. However, not all devices support all possible settings. This particularly makes the optimum network configuration more difficult for networks using twisted pair cables of the same kind and components which can be used with 10 Mbit/s or 100 Mbit/s in half duplex or in full duplex mode as desired. Imperfect configurations can lead to link errors or at least to performance losses because the maximum possible transmission rate is not used.

Due to this, the auto negotiation functionality (in the past also called Nway) has been established with the introduction of Fast Ethernet. With this functionality, the stations agree on the highest possible transmission rate and, if possible, full duplex operation. Then, all subscribers on the network configure themselves optimally.

However, problems could arise if one component in one segment is configured manually, i.e. if it has been set to a fixed transmission rate and mode and the auto negotiation function has been switched off. In this case, a device operating in auto negotiation mode informs the manually configured device about its possible settings but does not receive any response.

1.2.5 Ethernet and TCP/IP

Like nearly all standards in the field of data transmission, Ethernet also follows the ISO/OSI layer model. Based on this reference model, the principle course of a transmission is described. Each of the 7 parts (layers) has a particular function and makes it available for the next higher layer.

The Ethernet standard IEEE-802.3 defines the function of the two lowest layers. These layers consist of the following components and the Logical Link Control (LLC) which is described in the IEEE standard 802.2.

- Media Access Control Protocol (MAC)
- Physical Layer Signalling (PLS)
- Attachment Unit Interface (AUI)
- Medium Dependent Interface (MDI)
- Physical Medium Attachment (PMA)

Since the ISO/OSI model did not yet exist when the development of TCP/IP protocols started, these protocols are based on the DoD architecture. The DoD model cannot be clearly transferred to the ISO/OSI model. The following figure shows a comparison of Ethernet in the ISO/OSI model and the TCP/IP protocols adapted to that model. This shall explain that Ethernet does not necessarily mean TCP/IP (and vice versa). To be precise, TCP/IP is only based on Ethernet and can also be used in other data networks (e.g. for satellite links). In return, TCP/IP is not the only protocol used in Ethernet networks. Actually, TCP/IP is only one of numerous protocols which are used side by side.

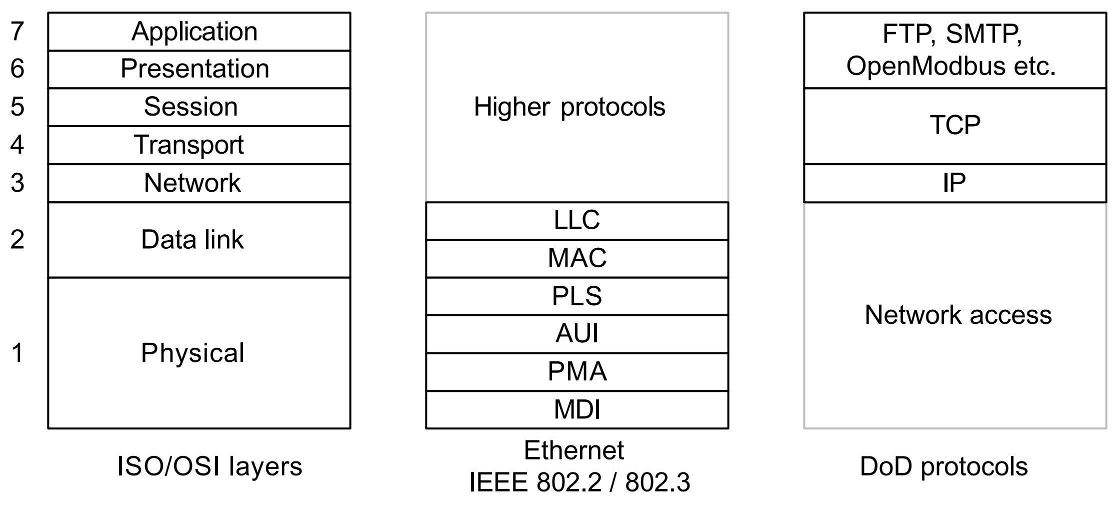


Figure 1-2: Ethernet in the ISO/OSI model

1.3 Protocols and applications

Section 1.2.5 Ethernet and TCP/IP only describes the context between Ethernet and TCP/IP. But actually numerous protocols are used in parallel in Ethernet networks. Furthermore, the protocols described here are not tied to Ethernet, as shown in the following figure.

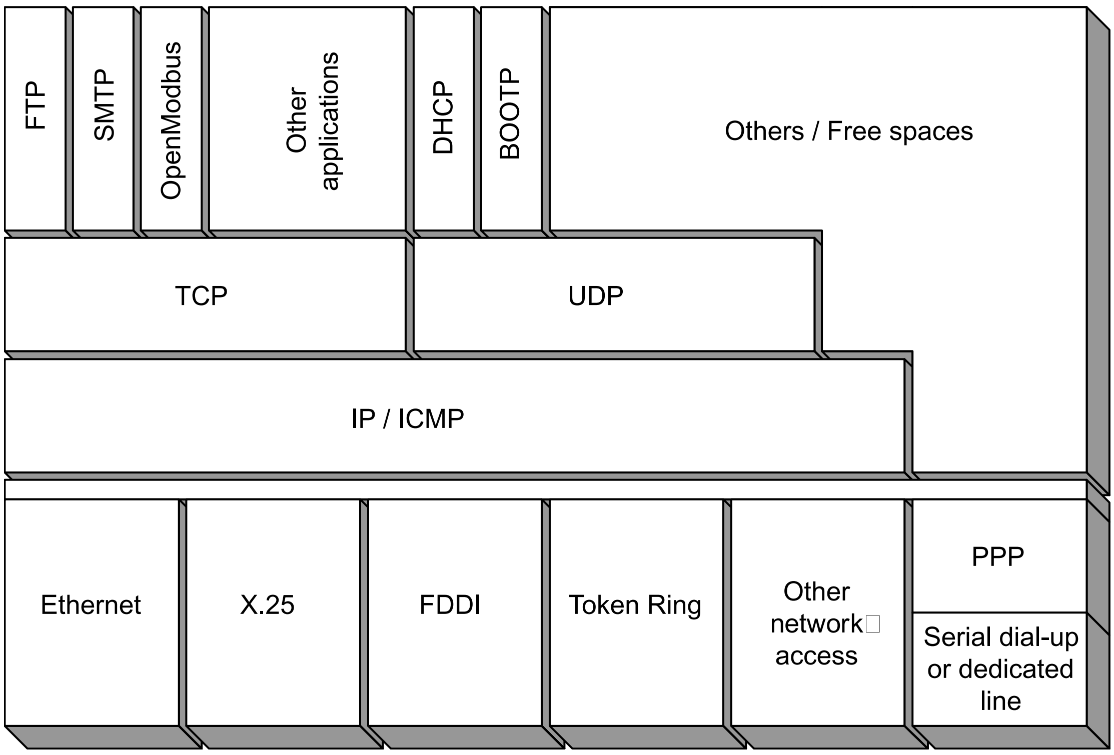


Figure 1-3: Ethernet protocols

1.3.1 Point-to-Point-Protocol (PPP)

PPP is a protocol for the transmission of LAN protocols (e.g. IP) via wide area networks. Thus, the point-to-point protocol enables the transmission of data via synchronous and asynchronous dial and leased lines. Since PPP is working independent of the corresponding physical interface, it can be used for modem connections as well as for ISDN connections or on RS232.

1.3.2 Internet Protocol (IP)

With the TCP/IP protocols, the Internet protocol takes on the tasks of the network layer. IP transmits the data packages of the higher protocol layers (incl. their headers) in the form of so-called datagrams via the network. The transport mechanism used for IP is connectionless, which means that the IP module of a transmitter does not verify whether the recipient actually received the data. This task has to be performed by higher protocol layers (e.g. TCP). Through the internet protocol, each datagram is transmitted as a single individual data package. During transmission, an independent route of transport is determined for each datagram in a network. Here it can happen that IP datagrams pass themselves on their way to the recipient and correspondingly arrive in a changed order. The task to sort the packages into the correct order is left to the protocols on the transport layer.

If a protocol of a higher layer passes data to be transmitted to the IP protocol, the IP protocol first generates a datagram consisting of the data to be passed and the IP header. If the target station is located in the local network, IP directly transmits the datagram to this station. If the target is located in a remote network, the datagram is first transmitted to a router in the local network which forwards the datagram to the target station, if necessary via further routers. During this process, each router involved in the transport of the datagram generates a new IP header.

Because in today's complex networks often several routes of transport are available between sender and recipient, several subsequent telegrams are not necessarily routed via the same way. Furthermore it can occur during the transport between sender and recipient that data packages are routed via networks with a maximum permitted telegram length smaller than the length of the datagram to be transmitted. In this case, the adjacent router splits the datagram into smaller data packages. This process is referred to as fragmentation. The individual fragments are provided with an own IP header and forwarded as independent data packages which can be routed to the target station via different ways and therefore also in a different order. The target station then has to assemble the data fragments to one single datagram before this datagram can be passed to the next higher layer. This process is called re-assembly mechanism.

IP header:

Each datagram transmitted via IP has a preceding IP header. Among other things, this header contains the IP addresses of the sender and the recipient, a checksum and a time to live identifier. The following figure shows the principle structure of an IP header.

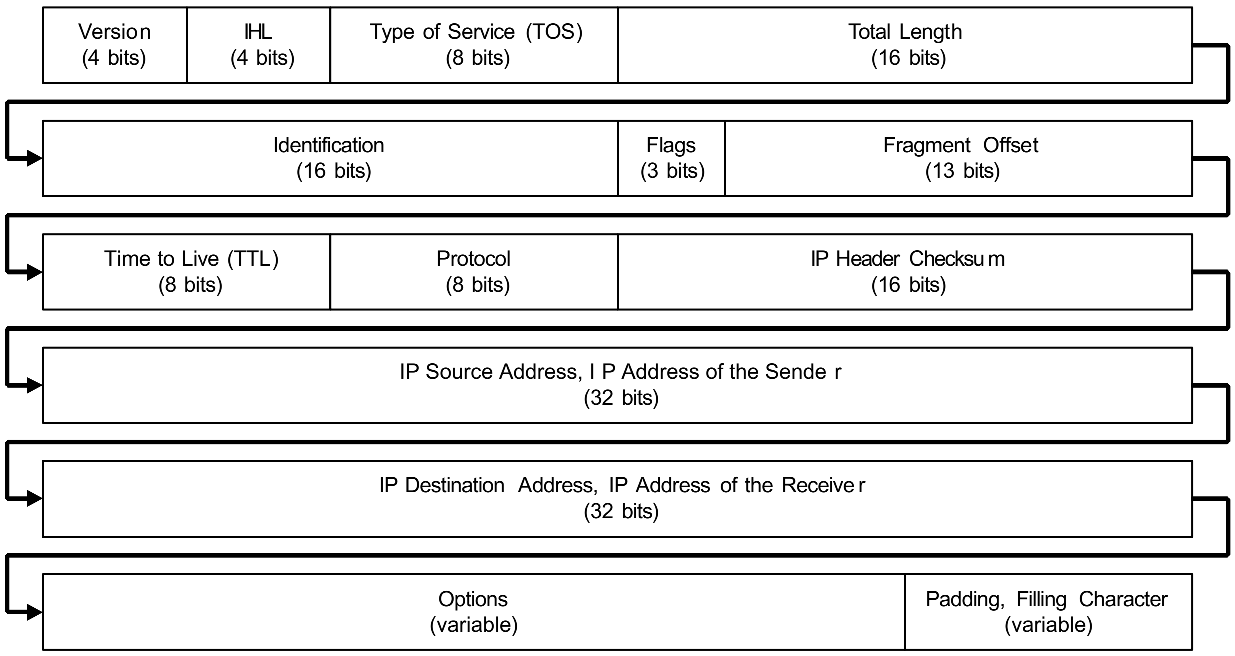

Version (4 bits)
IHL (4 bits)
Type of Service (TOS) (8 bits)
Total Length (16 bits)
Identification (16 bits)
Flags (3 bits)
Fragment Offset (13 bits)
Time to Live (TTL) (8 bits)
Protocol (8 bits)
IP Header Checksum (16 bits)
IP Source Address, IP Address of the Sender (32 bits)
IP Destination Address, IP Address of the Receiver (32 bits)
Options (variable)
Padding, Filling Character (variable)

Figure 1-4: Principle structure of an IP header

The total length of an IP header is defined in the IHL field (Internet Header Length) in numbers of 32 bit units (4 octets) and is, due to the variable size of the options field, 20 bytes or longer (5 x 4 octets = 20 bytes).

The following figure explains that an IP datagram embedded in the Ethernet frame reduces the actually transmittable user information. Further reductions result from the headers of the higher layers.

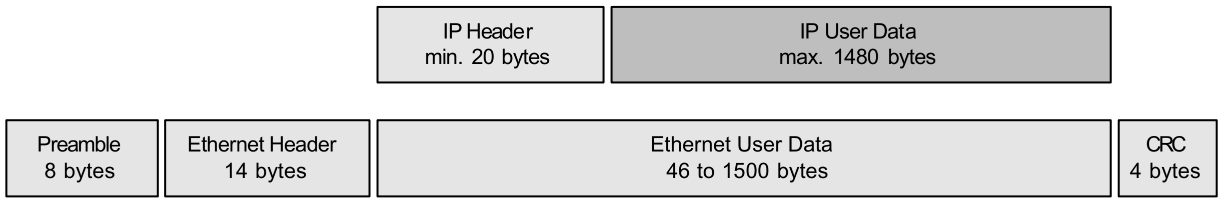


Figure 1-5: IP datagram in the Ethernet frame

Detailed descriptions of the content and meaning of the individual fields of the IP header are not given here, since such detailed knowledge is not necessary for normal application. Only the meaning of the checksum and the time to live identifier (TTL) as well as the structure of the IP addresses are described in the following sections.

The checksum is used by the recipient of an IP datagram to verify that the data package has been transmitted without errors. Datagrams with a faulty checksum are dismissed by the routers involved in routing the datagram as well as by the final recipient.

Each time a router receives an IP datagram, it reduces the time-to-live timer of the datagram by 1. If the value reaches 0, the datagram is destroyed or no longer forwarded. This shall prevent datagrams from endless straying in the network.

IP addresses:

Each station using the IP protocol must have at least one IP address. The IP address must not be confused with the Ethernet MAC address (refer to 1.2 Frame formats). An IP address is a 32 bit value which is normally represented using the 'dotted decimal notation' (e.g. 89.129.197.1).

An IP address is divided into two parts: the network address and the computer's address (node address). The network address part includes up to three bytes. The size of the network address, which determines the address class, is defined by the first four bits of the first octet of the IP address. The following table gives an overview of the various classes with their identification and use.

Class	**1st octet**	**Use**
Class A	000 - 127	Few networks, many computers
Class B	128 - 191	Medium distribution of networks and computers
Class C	192 - 223	Many networks, few computers
Class D	224 - 239	Multicast addresses
Class E	240 - 255	not defined

Table 1-1: Classes of IP addresses

Class A addresses consist of one octet for the network address and three octets for the computer address. For example, the IP address 89.129.197.1 designates computer 129.197.1 in network 89. Since the highest bit of the network address is always 0 and the IP addresses 0 and 127 are of particular significance, only 126 networks can be addressed by a class A address. Using the three octets left for the computer address, up to 16.777.216 computers can be addressed within these networks. Value 255 is reserved in each octet of the IP address for the broadcast address.

Class B addresses consist of two octets for the network address and two octets for the computer address. For example, the IP address 129.89.197.1 designates computer 197.1 in network 129.89. Since the two highest bits of the network address always have a value of 10, the address range reaches from 128 to 191. The second octet is also interpreted as a part of the network address. This way, 16.384 networks with up to 65.536 nodes each can be addressed with the class B address. Value 255 is reserved in each octet of the IP address for the broadcast address.

Class C addresses consist of three octets for the network address and one octet for the computer address. For example, the IP address 197.129.89.1 designates computer 1 in network 197.129.89. The three highest bits of the network address always have a value of 110. Consequently, the values 192 to 223 are valid in the first octet and the values 1 to 254 in the subsequent two octets. This results in an address range of 2.097.152 networks with up to 254 nodes each. Value 255 is reserved in each octet of the IP address for the broadcast address.

The range 224 to 239 (class D) is intended for multicast addressing as it is frequently used today for newer protocols of the IP protocol family.

Class E addresses are reserved for future use.

The following table gives an overview of the IP address structure for the individual classes.

Network address	1st octet								2nd octet								3rd octet								4th octet							
	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
Class A	0	Network address							Computer address																							
Class B	1	0	Network address														Computer address															
Class C	1	1	0	Network address																					Computer address							
Class D	1	1	1	0	Multicast address																											
Class E	1	1	1	1	Undefined format																											

Table 1-2: Overview of IP address structures

IP address assignment:

The allocation of IP addresses or the assignment of an IP address to a device can be done in different ways (see also 2 The Ethernet coupler).

The simplest case is the static assignment of the IP address. Here, the device is configured with a fixed address which is assigned by the network administrator. Since the IP address always includes the identification of the network the device is used in, the network administrator has to be informed about the device's place of installation. One disadvantage of this method appears, if the device shall be later used in another network. In this case, an IP address has again to be requested from the administrator and set manually. The advantage of this method is that the IP address of the device is known and that the device can always be called under this address.

Alternatively the IP address can also be obtained from a BOOTP or a DHCP server (refer to 1.3.7 BootP and DHCP). In this case, the IP address is assigned on the basis of the device's MAC address when switching on or booting the device. Here, we have to distinguish static and dynamic assignment. The advantage of both methods is that no further settings have to be done for the device. This also applies if the device is later used in another network. The disadvantage of the dynamic method is that the IP address of a device can possibly change when it is switched on again the next time. Therefore, it is not possible to implement a fixed communication connection to another device (e.g. two controllers via function blocks).

For all methods mentioned above, fixed implemented connections to other devices have to be implemented again if the IP address of the device has changed. In the first two cases (direct address configuration on the device and automatic static assignment), this will normally not be necessary since the IP address of the device has only to be changed if the device shall be used in another network. In the latter case (dynamic IP address assignment), this problem is not only restricted to inter-network connections since the IP address of a device can change each time the device is switched on, even if both devices involved in the connection are located within the same network or if their places of installation do not change.

Special IP addresses:

As already mentioned in the section before, special IP addresses with a special function exist. These addresses are concisely described below.

127.x.x.x

The class A network address with the number 127 is reserved for the loopback function of a station, independent of its network class. By definition, all IP addresses beginning with 127 may only be used for internal testing. They must not be transmitted via the network. Datagrams addressed to network address 127 are returned to the sender within the TCP/IP software (loop). This allows testing of the software function. The network controller itself is not tested this way. If the network controller shall be included in the test, it is recommended to use the ping application.

Value 255 in an octet

The value 255 is reserved as a broadcast address (all one broadcast). To address all computers within the same network, the computer's part of the network address has to be set to 255 (e.g. 91.49.1.255). The use of subnet masks possibly complicates the calculation of the broadcast address since in this case an extended algorithm has to be used.

Value 0 in an octet

There are two definitions for the use of the value 0 within an IP address. An earlier definition intended value 0 as a broadcast. This all zero broadcast should no longer be used today. Now, the all one broadcast (see above) should be used instead. The second definition for the address value 0 which is presently valid allows the identification of the own network (this net) or computer (this host). For example, an IP address of 0.1.1.1 means that computer 1.1.1 in this network shall be called.

Subnet mask:

When configuring a device, which is connected to a TCP/IP network, it is often possible not only to set the IP address but also to preset a so-called subnet mask. If the IP address of the device is configured manually, the subnet mask has also to be entered manually, if it is used. If the device is set to automatic IP address assignment (refer to sections "IP address assignment" and "1.3.7 BootP and DHCP"), the assignment of a possibly required subnet mask is also performed automatically.

The use of subnet masks allows network administrators to divide a large network into several small (sub) networks. Subnetworks are often used to image the topological structure of large networks (e.g. the departments or floors within one building) or to decentralize the administration of the network. Furthermore, logical transitions between different transfer media can be easily implemented this way. An IP router is able to connect different physical networks to each other. The only condition for this function is the existence of an own network address for the router in each connected network. A subnet is only known by the computers connected to the local network. Outside of the local network, such an address appears as a usual IP address.

With this method which is called 'subnetting', specific bits of the computer's IP address part are used to expand the class-specific network address part. This reduces the address range available for the addressing of computers in the subnet. The IP address is then divided into the network address, the subnet number and the computer's address. The subnet number is defined by means of a bit mask (subnet mask). With this, the IP address and the subnet mask are bit-wise logically interconnected by an AND. Consequently, the subnet mask defines which parts of an octet are to be interpreted as the (sub) network or as the node address. As a result, the software of the IP protocol distinguishes devices connected to the local subnet and devices located in other subnets. The computer is located in the local network if the result of the AND interconnection of the IP address and the subnet mask delivers the local network address and the local subnet number. Otherwise the datagram is routed to another subnet router.

For example, by applying a subnet mask of 255.255.0.0, a class A address (e.g. 126.x.y.z) with a default subnet mask of 255.0.0.0 becomes a class B address and by applying a subnet mask of 255.255.255.0, this address becomes a class C address. The use of subnet masks is explained in the following examples.

Example 1:

Computer A with IP address 89.236.4.85 and subnet mask 255.224.0.0 wants to establish a connection to computer B with IP address 89.234.85.50.

IP address computer B (bin)	01011001	11101010	01010101	00110010
Subnet mask computer A (bin)	11111111	11100000	00000000	00000000
AND interconnection (bin)	01011001	11100000	00000000	00000000

The result corresponds to the IP address of computer A expanded by the subnet mask. Consequently, computer B is located in the local subnet and can be addressed directly.

IP address computer A (bin)	01011001	11101100	00000100	01010101
Subnet mask computer A (bin)	11111111	11100000	00000000	00000000
AND interconnection (bin)	01011001	11100000	00000000	00000000

Example 2:

Computer A with IP address 89.236.4.85 and subnet mask 255.224.0.0 wants to establish a connection to computer C with IP address 89.211.1.22.

IP address computer C (bin)	01011001	11010011	00000001	00010110
Subnet mask computer A (bin)	11111111	11100000	00000000	00000000
AND interconnection (bin)	01011001	11000000	00000000	00000000

The result does not correspond to the IP address of computer A expanded by the subnet mask (refer to example 1). Consequently, computer C is not located in the local subnet. The datagram has to be transferred to computer C via a router.

Example 3:

A network with the class A address 126.x.y.z can be divided into two subnets by means of the subnet mask 255.128.0.0 where one subnet covers the range from 126.0.y.z to 126.127.y.z and one the IP address range from 126.128.y.z to 126.255.y.z. For the broadcast addresses of the subnet masks the addresses 126.127.255.255 and 126.255.255.255 respectively are automatically used since in these cases all bits of the host part of the address are set to 1 (all one broadcast).

Gateway:

Another parameter that can be set in the IP software of some devices is the gateway or standard gateway parameter. This parameter defines the IP address of the gateway (router) to which datagrams addressed to a computer outside of the local network are to be transferred. If the IP address of the device is configured manually, the IP address of the gateway has also to be entered manually, if it is used. If the device is set to automatic IP address assignment (refer to sections "IP address assignment" and "1.3.7 BootP and DHCP"), the assignment of a possibly existing gateway is also performed automatically.

1.3.3 Internet Control Message Protocol (ICMP)

Aside from the main protocol IP, the auxiliary protocol ICMP also exists on the network layer. ICMP is used to exchange information and error messages between the network subscribers. The ICMP protocol is based on the IP protocol and is thus treated by IP like a protocol of a higher layer. ICMP data are always transferred together with a complete IP header. The actual ICMP messages are included in the subsequent IP data part.

The following information messages are defined in the ICMP protocol:

- Echo
- Information
- Address mask / address format
- Router discovery

Echo request/reply message

The recipient of an echo request datagram returns all data contained in the received data package to the sender by means of an echo reply datagram.

The best-known tool based on ICMP echo packages is the ping command. Ping (Packet Internet Gopher) is implemented in virtually all IP terminal devices. It is based on the principle that one device sends an ICMP echo request and then waits for the response. Depending on the implementation, only a success message (host alive) or a failure message (no response from host) is output or even the wait time for the response.

Information request/reply message

By means of an information request message, the sender of a datagram is able to poll the network address of the network it is connected to. For this purpose, it sends a corresponding request to IP address 0 and then receives a reply message with the local network address from any device connected to the network.

Timestamp request/reply message

In a timestamp request, the sender transfers its local time as the sending time in the datagram to the recipient. The recipient of the request then repeats the sender's sending time in its reply message to the sender and supplements this by its local reception time of the request and the local sending time of the reply.

Address format request/reply message

By means of an address format message, the sender of a datagram is able to poll the subnet mask length or address as bit values from another subscriber in the network.

Router discovery message

By means of the router discovery messages, a device in the network is enabled to automatically determine the IP addresses of the connected routers (gateways). It is distinguished between router advertisement messages and router solicitation messages. With router advertisement messages a router periodically transmits the availability of its network interfaces by multicast. In contrast to this, router solicitation messages are sent by a network node to one or more routers after its initialization. The routers that received this message then return one single router advertisement message.

Furthermore, the following ICMP error messages exist:

- Destination unreachable
- Redirect
- Source quench (resources exhausted)
- Time exceeded
- Parameters problem

Using the ICMP error messages, the sender of a datagram is informed about the reason why the datagram could not be transmitted.

Destination unreachable message

A destination unreachable message can have different causes. For example, it is possible that the higher protocol on the target computer is unknown or that the called destination port is already in use. Interruptions of the network can also cause destination unreachable messages.

Redirect message

Using a redirect message, the sender of a datagram is informed via which router the datagram can be transmitted to the target network. This message is generated by a router if it receives a datagram which is addressed to a target network that cannot be reached via this router. However, the router knows the IP address of the correct router and informs the sender about this address using a redirect message.

Source quench message

A source quench message is for example output by a target computer if it is not able to process the amount of received data fast enough. Using this message, the sender is requested to reduce the data transfer rate until no more source quench messages are received.

Time exceeded message

One reason for a time exceeded message could be for example that the IP protocol is not able to re-assemble a fragmented datagram within a defined period of time to a full data stream during the re-assembly process.

Parameter problem message

For example, a parameter problem message is issued if a router detects an error in the IP header of a datagram (e.g. an incorrectly set argument in the IP options) and consequently destroys the datagram or doesn't forward it.

1.3.4 Transmission Control Protocol (TCP)

The transmission control protocol implements the tasks of the transport layer in the ISO/OSI model and therefore is directly based on the IP protocol. TCP works connection-oriented. It makes connection services available for an order-accurate and secure transmission of user data. TCP enables the detection of data losses, the automatic re-transmission of lost data packages as well as the detection of duplicates. Compared with TCP, connectionless operating protocols of the transport layer which do not support such mechanisms (e.g. UDP) provide higher performance.

TCP treats the data handed over by the higher protocols as one quasi-continuous data stream, segments these data and then transmits them in single data packages. On the receiver's side, the single segments are re-assembled again and made available for the higher protocols as a data stream. The TCP protocol is completely independent of the individual network-specific properties. Hence, a TCP segment can have a length of up to 65 kB. When data are transmitted via a network, the lower IP protocol has to fragment the TCP segments further into several small packages of a low order according to the properties of the network (e.g. Ethernet). The following figure shows the course of the transfer processes.

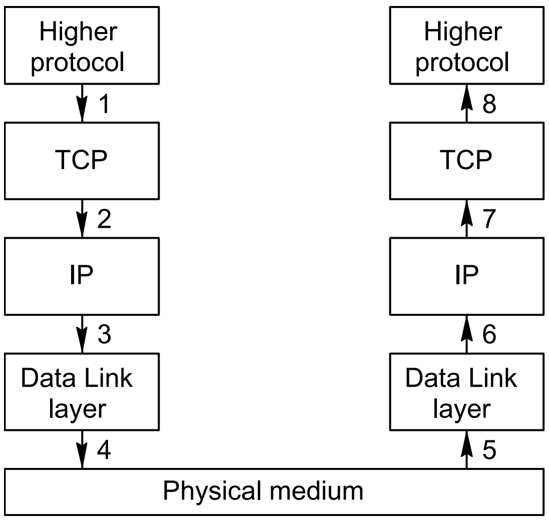


Figure 1-6: Course of the transfer processes

Port:

The multiplex mechanism of the transmission control protocol (TCP) allows the simultaneous use of TCP by a large number of higher protocols and application processes. The assignment of the individual data streams to the above applications is done via so-called ports. For the access to specific standard applications firmly defined port numbers exist (well-known ports). These port numbers range from 0 to 1023 and are assigned by the IANA (Internet Assigned Numbers Authority). All other ports from 1024 to 65535 are not under the control of the IANA and can be used as desired.

In the following table some well-known ports are listed.

Application	**Port number**
FTP data (FTP data channel)	20
FTP control (FTP control channel)	21
Telnet	23
SMTP (Simple Mail Transfer Protocol)	25
DNS (Domain Name Server)	53
http (HyperText Transfer Protocol)	80
OpenModbus	502

Table 1-3: Well-known ports

Socket:

For the purpose of unique identification of a TCP application process, the port number and the IP address of a device are locally summarized to a so-called socket. A socket with a firmly defined port number is designated as well-known port. The following simplified example explains the principle of the logical connections between pairs of sockets.

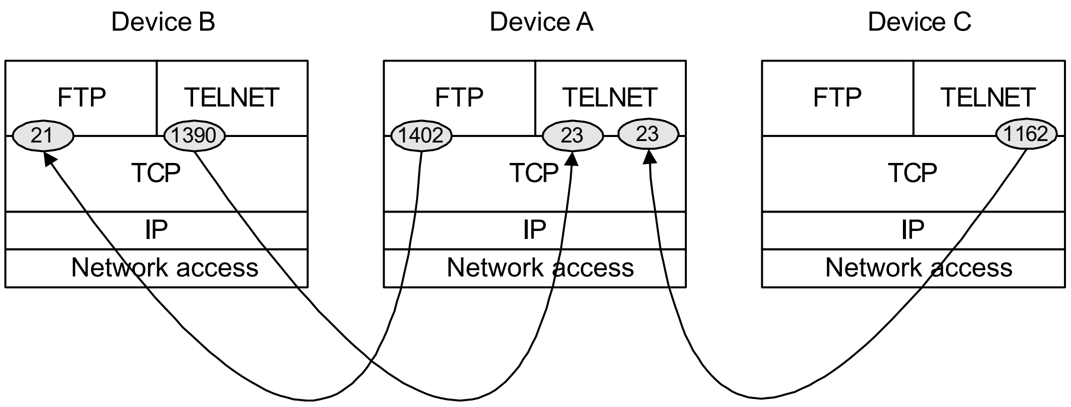


Figure 1-7: Principle of the logical connections between pairs of sockets

The devices A, B and C are located within the same network. Each device has a unique IP address. The individual TCP modules have several independent ports with unique port numbers.

Device A wants to exchange files with device B via FTP and establishes for this purpose a connection from its local port 1402 to the well-known socket 21 of device B. For this, the local port number of the device that initiated the connection (here: device A) is not of importance because the corresponding application is already uniquely defined by the active connection establishment. The connection establishment only requires that the request can be identified as FTP control in device B through the use of the socket or port 21.

At the same time device B dials in (from local port 1390) to device A. There it uses port 23 (Telnet). Since it's a telnet access, the well-known socket 23 is several times available in device A. Therefore, device C is able to establish a parallel telnet connection to device A via its local port 1162.

After the respective logical connection has been established successfully, all application data of the respective active device to be transferred are disassembled into TCP segments and transferred to the corresponding port of the respective remote device. Each port of the TCP module of a remote device accepts and sends only data from and to the TCP module of the corresponding active device. In our example, device A has only two telnet ports. This is why every request for another connection to this port is denied as long as both connections are open. Once all data have been exchanged within the bounds of a connection, the respective connection is closed and the corresponding port is available again for new requests.

TCP header:

Each data package transmitted via TCP has a preceding TCP header. Among other things this header contains the respective ports of the sender and the recipient. The meaning of the term ports is explained earlier. Detailed descriptions of the further components of the TCP header and their functions are not given here, since such detailed knowledge is not necessary for normal application. The following figure shows the principle structure of a TCP header.

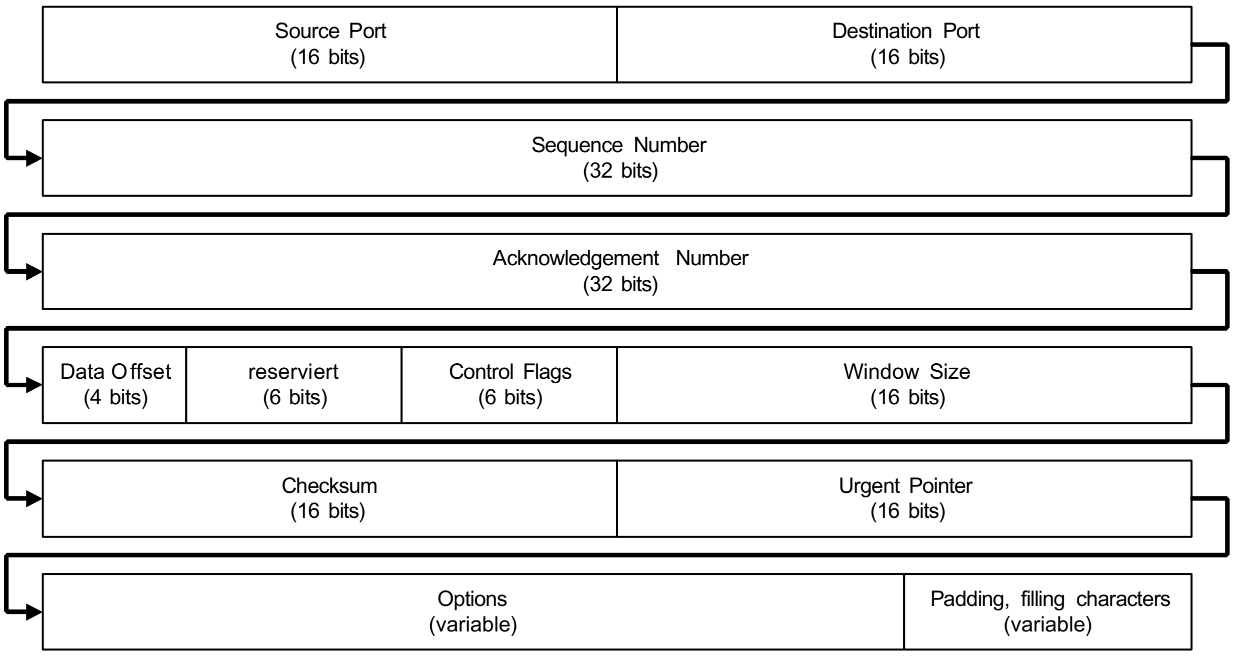


Figure 1-8: Principle structure of a TCP header

The total length of the TCP header depends on the type and number of options and is at least 24 bytes.

The following figure explains that a TCP segment embedded in an IP datagram and then packed in the Ethernet frame reduces the actually transmittable user information. Further reductions result from the headers of the higher layers.

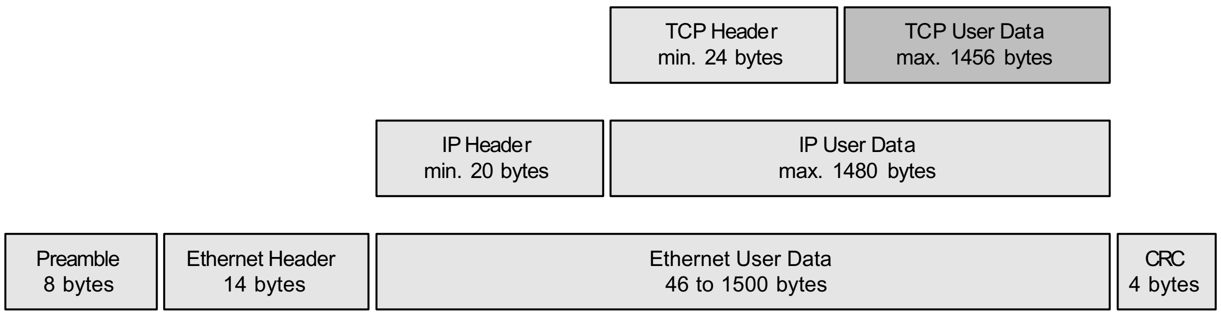


Figure 1-9: TCP segment in the Ethernet frame

Phases of a communication:

In the previous sections, it was only explained that TCP connections are established first, then guarantee a safe transmission and finally are closed again after transmission is finished. For a better understanding of these processes in the TCP protocol, the context is explained in more detail below. The following figure shows a simplified state diagram of a TCP module.

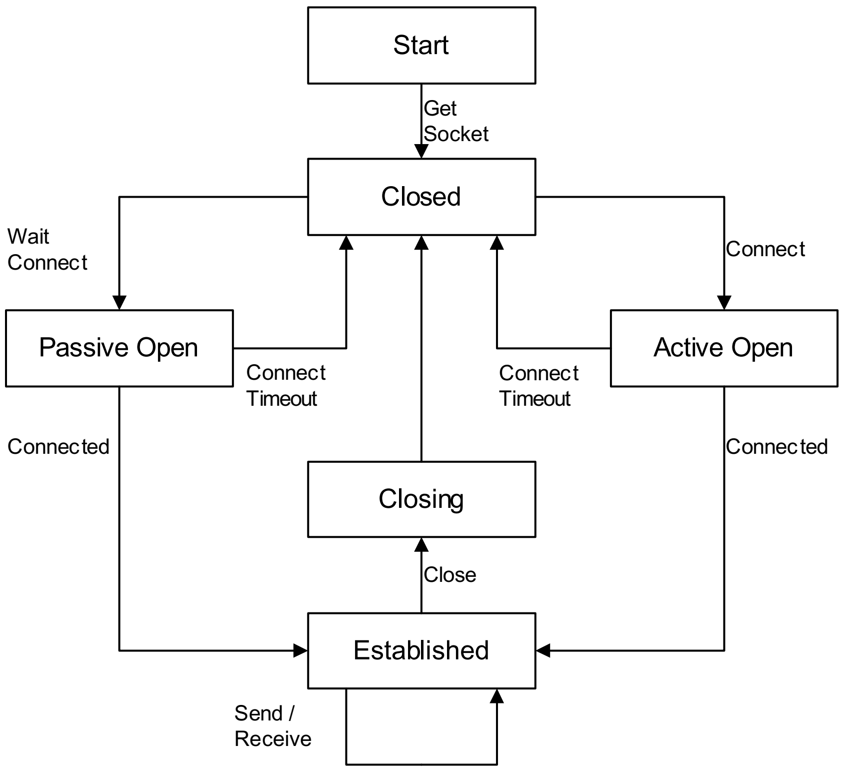


Figure 1-10: Simplified state diagram of a TCP module

In the figure above it is assumed that a device is switched on or that an application is started (state: start). The first TCP action starts with the request for a socket. After a successful assignment of a socket through the TCP module, the actual TCP starting point is reached. Starting from this state, there are two possibilities to open a connection, in active mode or in passive mode. In active mode, an active connection establishment to a network node is tried (Connect) whereas a device in passive mode waits for an incoming connection request (Wait Connect). Between the active and the passive device, first various connection parameters are negotiated. Then the connection is established (Established) and data can be exchanged between the higher protocols (Send / Receive). After all data have been exchanged, the disconnection (Close) is initiated. After a further exchange of connection parameters (Closing), the connection is closed and the TCP module is again set to the Closed state.

Aside from different safety mechanisms, each TCP state has **timing supervision functions**. Since some of these timers can be parameterized by the application or by the user, we want to briefly explain their functions below.

Connect Timeout

The connect timeout function determines the time period after which an attempt of a connection establishment shall be aborted. In active mode, setting this time period is obligatory, i.e. a finite time value has to be preset for the connection setup. If the TCP module cannot establish the specified connection within this time, the attempt is aborted with an error message. Possible reasons for this could be for example that the addressed node is not located inside the network or that no corresponding socket is in the Wait Connect / Passive Open state.

In passive mode, presetting of a connect timeout value is optional. If a time period is specified, the list state (Passive Open) is left with an error message if no attempt for a connection setup to the local socket occurred during this period. If no time period is specified, TCP waits for an incoming connection for an unlimited time.

Retransmission Timeout

Following a successful connection establishment the TCP data transmission phase takes place. The reception of a data package is confirmed to the sender by a positive acknowledgement (ACK) of the recipient. Due to the flow rate, only one single ACK message is used to confirm the reception of a whole series of packages instead of confirming each single data package. The confirmations of reception are monitored on the sender's side using the so-called retransmission timer. One retransmission timer is started each time a data block is sent. If (e.g. due to a transmission error) no confirmation is received before the timer has elapsed, the package is sent again.

Give-Up Timeout / Send Timeout

The give-up timer is also used for timing monitoring of the reception confirmations for data segments sent by the recipient. After this time period, the connection is closed if no data segments have been confirmed by the target node.

Inactive Timeout

The inactive timeout is used to monitor the activity on a connection and is normally implemented by the application. The timer is newly started with each data reception. If no data packages are received before this timer has elapsed, the connection is automatically closed. The use of the inactive timeout function is optional. The inactive timeout cannot be used together with the keep alive mechanism (described below).

Keep Alive Timeout

The keep alive mechanism can be used to test whether a connection that did not carry data over a longer period of time still exists. The timer is newly started with each data reception. After this period of time, the keep alive timer generates data packages (so-called keep alive probes) which are sent to the other side of the connection. If these packages are not confirmed, the connection is closed. The use of the keep alive timeout function is optional. The keep alive mechanism cannot be used together with the inactive timeout (described above).

Close Timeout

A TCP connection is closed after data transmission is finished. There are two possible methods for closing a connection, a graceful close or an abortion. Graceful close describes the normal and coordinated closing of a connection. Both communication partners come to an agreement about closing the connection. Optionally, this phase can be monitored by defining a close timeout value. The respective timer is started with the request for a connection establishment. If the connection cannot be closed in a coordinated manner within the specified time period, the connection is aborted (hard close). A close timeout value of 0 results in an immediate abortion of the connection. An abortion represents the closing of a connection forced by one side. This can lead to a loss of all data on the communication connection.

1.3.5 User Datagram Protocol (UDP)

In addition to TCP, the user datagram protocol is a second protocol implemented on the transport layer (layer 4). It is also directly based on the IP protocol. In contrast to TCP, UDP is operating connectionless and not connection-oriented. It does not provide end-to-end control. Transmission via UDP does not guarantee the correct delivery of a datagram to the recipient, the detection of duplicated datagrams and the order-accurate transmission of the data. If necessary, this has to be guaranteed by suitable mechanisms on the application side.

Compared to TCP, UDP has the advantage of a considerably higher performance. This is particularly reached by the missing safety mechanisms. No logical connections are established. The individual datagrams are treated as completely independent events instead. This particularly becomes apparent for multicasts or broadcasts. In case of TCP, first numerous connections have to be established for this, then the data have to be transmitted and finally the connections have to be closed again. For UDP this is restricted to the pure data transmission. In the end, the functions mentioned before result in the fact that the UDP header is considerably shorter than the TCP header and thus causes less protocol overhead.

Port:

Like TCP, UDP provides several ports (access addresses) to allow simultaneous access to the protocol for several processes (see also "Port" in section "1.3.4 Transmission Control Protocol (TCP)"). The assignment of the port numbers to the application processes is performed dynamically and optional. However, fixed port numbers (assigned numbers) are defined for specific frequently used standard processes. The following table shows some of these assigned numbers.

Application	Port number
Reserved	0
Echo	7
IEN 116	42
Domain Name Service	53
BootP	67/68

Table 1-4: Fixed port numbers (assigned numbers) for standard processes

Socket:

For unique identification of a UDP application process, the port number and the IP address of a device are locally summarized to a so-called socket. A socket with a firmly defined port number is designated as well-known port. A pair of sockets uniquely identifies a pair of processes communicating with each other (see also the example under "Socket" in section "1.3.4 Transmission Control Protocol (TCP)").

UDP header:

Each data package transmitted via UDP has a preceding UDP header. This header is considerably shorter and simpler than the TCP header and only contains the source and destination codes (port number of the source in the sender and the target in the recipient) as well as the total length of the data package and a checksum. The UDP user data immediately follow the header. The following figure shows the principle structure of a UDP header.

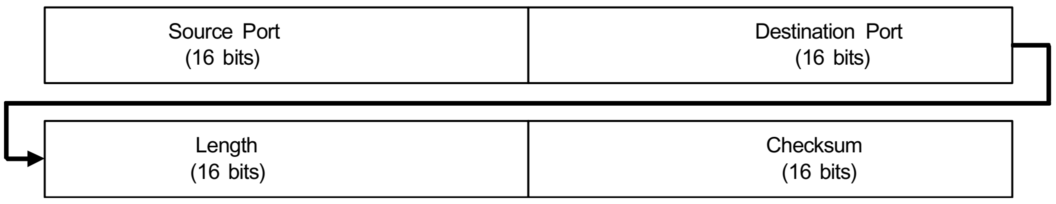


Figure 1-11: Principle structure of a UDP header

The UDP header has a length of only 8 bytes. The calculation of a checksum is optional. A UDP checksum value of 0 means for the recipient that no checksum has been calculated in the sender.

The following figure shows a comparison between a UDP and a TCP data package, both embedded in an IP datagram and then packed in the Ethernet frame. Further reductions of the maximum possible user data length result from the headers of the higher protocol and application layers.

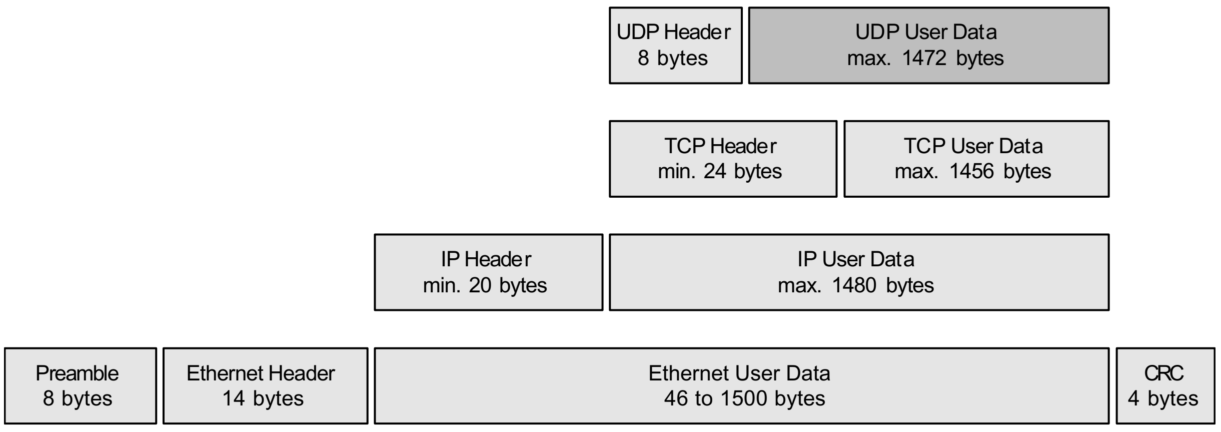

UDP Header 8 bytes
UDP User Data max. 1472 bytes
TCP Header min. 24 bytes
TCP User Data max. 1456 bytes
IP Header min. 20 bytes
IP User Data max. 1480 bytes
Preamble 8 bytes
Ethernet Header 14 bytes
Ethernet User Data 46 to 1500 bytes
CRC 4 bytes

Figure 1-12: Comparison between UDP and TCP data packages

1.3.6 OpenModbus on TCP/IP

OpenModbus is a protocol of the application layer above TCP/IP. The basis for (Open-)Modbus is provided by the Modbus RTU specification. Modbus RTU is a standardized master-slave protocol for serial transmission lines which allows the master to read and write bit (coil) and word values (register) from/to the slaves (see also reference to the existing documentation "Modbus Telegrams").

The major field of application for the Modbus protocol is the fast data exchange between the automation devices as well as between visualization systems (HMI / SCADA) and automation devices.

OpenModbus on TCP/IP uses port 502 by default.

Telegram structure:

Modbus telegrams are composed of a transmission path-independent telegram part (simple Protocol Data Unit, PDU) and a network- or bus-specific header. The PDU and the header together are the application data unit (ADU). The following figure shows the general structure of a Modbus telegram as well as the structure of an OpenModbus on TCP/IP telegram.

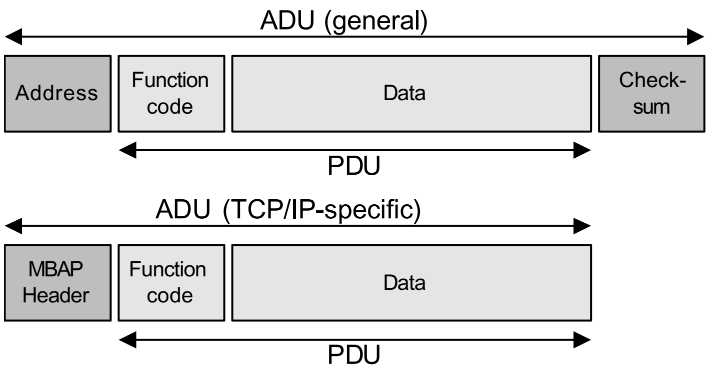


Figure 1-13: Structure of an OpenModbus on TCP/IP telegram

The telegram structure differs in some points from the original Modbus RTU format for serial transmission:

- The slave address at the beginning of the telegram is replaced by the 'Unit Identifier' field in the MBAP header (see below).
- The telegrams are structured in a way that the recipient is able to clearly identify their size. For some function codes the telegram length is fixed, so that it is possible to immediately determine the length from the function code. For function codes with variable telegram lengths, the data part of the telegram contains an additional length specification.
- Since the IP header as well as the TCP header and the Ethernet frame have their own checksums, the additional Modbus checksum is not required.

Modbus Application Protocol Header (MBAP)

The Modbus Application Protocol Header has a length of seven bytes and is structured as follows.

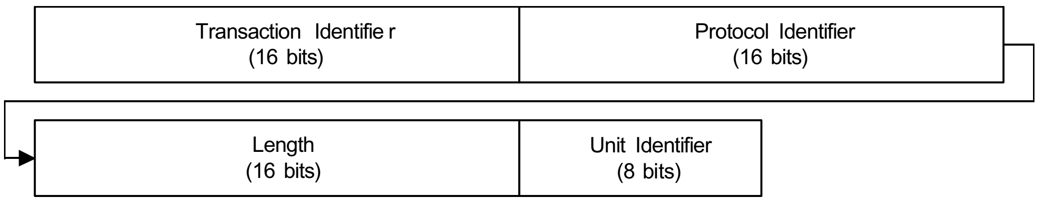

Transaction Identifie r (16 bits)
Protocol Identifier (16 bits)
Length (16 bits)
Unit Identifier (8 bits)

Figure 1-14: Structure of the Modbus Application Protocol Header

The 'Transaction Identifier' identifies a single process. The value is individually assigned by the master (client) on the occurrence of a request and copied to the respective response by the slave (server). This way the master is able to clearly assign an incoming telegram to the previous request even if there are several services running simultaneously.

The 'Protocol Identifier' is used to distinguish different future protocols. A Modbus master always sets this value to 0. A slave copies the value in the request to the header of the response.

The length field specifies the total length of the subsequent fields including the 'Unit Identifer' and the user data in bytes. The length is calculated by a master as well as by the slave when a telegram is generated.

The 'Unit Identifier' is used for routing purposes. Typically it is used for the conversion of a telegram by a gateway, if the master is located in a TCP/IP network and connected via this gateway to a serially connected Modbus or Modbus+ slave.

1.3.7 BootP and DHCP

The addresses on the lowest layers of a network arise from the used network technology. In case of Ethernet, these are the MAC addresses which are predetermined for each device. The IP addresses are located above layer 2. IP addresses are freely assigned depending on the network structure. By definition, each device within a TCP/IP network must have a unique IP address before it can send or receive data via the network. If a device is able to permanently store the IP address once it has been set, the protocol software is initialized with the stored IP address each time the device is booted later. If a device is not able to store its IP address or if it still has the delivery settings, the device is booted without a valid IP address. For such cases, protocols exist which allow the assignment of IP addresses via the network. The best-known representatives of these protocols, BootP and DHCP, are described in the following sections.

Bootstrap Protocol (BootP):

The bootstrap protocol is specified in RFC 951. As an application it is directly based on UDP. BootP works in accordance to the client-server principle. It uses port 67 for the communication with the server and port 68 for the communication with the client. The BootP client communicates with the server by exchanging one single data package. Aside from the IP address of the computer, this data package can additionally contain the IP addresses of the next router and a particular server as well as the name of the boot file. By means of another manufacturer-specific input-output map, additional information can be entered, for example the subnet address and the domain name server.

In the initial state, the client first has no valid IP address. To be able to send a request to the BootP server via IP, the client first uses the special IP address 255.255.255.255 (broadcast). Among other things, this request contains the MAC address of the client. The response from the BootP server is also sent as IP broadcast. By comparing the received MAC address with its own address, a client determines whether this data package is directed to it. This way the client is informed about his own IP address and other things. The assignment of an IP address (and the additional information belonging to that) to a MAC address is performed in the BootP server using static tables administered by the network administrator.

Since the Bootstrap protocol is based on UDP, the transmission safety mechanisms known from TCP are not available. BootP has to be able to detect possible errors on its own. Each time it sends a BootP request, the client starts a retransmission timer. If it doesn't receive a BootP reply before the timer has elapsed, it generates the data packages again. To avoid that all clients send their requests to the BootP server at the same time (e.g. after a voltage breakdown) and thus cause a server overload, the BootP specification recommends to use a random value for this timer.

Aside from the possibility to poll its own IP address, the bootstrap protocol can also be used by clients that already know their IP address to poll additional information, e.g. Name Server.

Dynamic Host Configuration Protocol (DHCP):

Today, many bigger companies have complex and sometimes global TCP/IP networks. The set-up and maintenance of such large networks is very expensive. Moves of individual employees or entire departments normally require to change the IP addresses and other parameters, e.g. the subnet mask. Furthermore already in the beginning nineties worries came up that a bottle-neck could arise in the assignment of free IP addresses. For these reasons the Dynamic Host Configuration Protocol (DHCP) was developed on the basis of the existing bootstrap protocol and published in 1993 in the RFC 1541. DHCP enables a central administration and maintenance of all TCP/IP configuration parameters by a network administrator and thus allows to build up a plug-and-play TCP/IP network. In contrast to BootP which only allows the static assignment of network addresses, DHCP supports three alternative ways of IP address assignment.

Automatic IP address assignment

With the automatic assignment, any fixed IP address is assigned to a client. On the initial login of the client to the server, a free address is assigned to the client. This assignment is stored in the server. Then, this IP address is assigned to the client each time it logs in to the server. This IP address cannot be used by another computer even if is not actually needed by the client, e.g. because the client is temporarily disconnected from the network. Compared with the method of setting the IP address directly on the device itself, this method has the disadvantage that for example a connection between two devices cannot be implemented until both devices have logged in to the DHCP server at least one time since the IP address is not known before.

Dynamic IP address assignment

The dynamic address assignment assigns an IP address to the client only for a specific period of time when it logs in to the server. The client can also release this address on his own initiative before the time is over if it doesn't need it any more. The advantage of this method is that an IP address which is no longer needed by a client can be assigned to any other client. In practice, this could for example be used for an office that has less IP addresses than employees and where always only a part of the employees is present. A disadvantage of this method is that no fixed connections can be implemented between network nodes using IP addresses since the IP address can change with each login of a client. In practice, this could for example be relevant for an installation with several controllers that exchange the process data via corresponding function blocks in the user program.

Manual IP address assignment

Using the manual assignment, the network administrator is able to explicitly assign an IP address to a client by defining corresponding static tables in the server. This means that the administrator already defines in advance which configuration he wants to assign to which client. In this case, DHCP is only used as a means of transport. A disadvantage is that the IP addresses cannot be used flexibly. An advantage arises from the fact that the IP addresses of all devices are known from the beginning and can be taken into account for the implementation of fixed communication connections already during planning phase.

Function

DHCP also supports the BootP relay agents defined in the bootstrap protocol. The BootP relay agents have the task to forward DHCP messages from and to network segments which do not have an own DHCP server. Therefore, relay agents to a certain extent carry out the functionality of a router on DHCP level. The use of relay agents has the advantage that not every subnet requires its own DHCP server.

The processes of the DHCP protocol are implemented using seven different types of messages.

Type of message	**Description**
DHCP Discover	Client broadcast to localize the available DHCP servers in the network.
DHCP Offer	Broadcast or unicast response of a server to a discover message with configuration parameters for the client.
DHCP Request	Broadcast of a client to all servers to request the offered parameters from a dedicated server with the simultaneous rejection of the parameters of all other servers.
DHCP Ack	Message from a server to a client, includes configuration parameters and the IP address.
DHCP Nak	Message from a server to a client, rejection of the request for configuration parameters and IP address.
DHCP Decline	Message from a client to a server saying that the configuration parameters and the IP address are invalid.
DHCP Release	Message of a client to a server saying that the IP address is no longer needed and available for use again.

Table 1-5: Message types of the DHCP protocol

Below the initialization process for a dynamic assignment is explained from the client's point of view.

After the boot process, the client is first in the initial state and sends a DHCP Discover message within its local subnet (subnet broadcast). This message can optionally already include a suggestion of the client for the required parameters. This can be the case if the client has already received corresponding values during a previous process and stored this. In any case, the client has to hand over the hardware address (MAC address) since at this point in time it is not yet clear whether the client already has a valid IP address and can be accessed via this address. Since not every subnet necessarily has its own DHCP server, the DHCP Discover message is also forwarded to further subnets via possibly existing relay agents.

After it has sent the DHCP Discover message, the client is waiting for configuration offers in the form of DHCP Offer messages from DHCP servers that are waiting for its request. The servers try to directly respond to the client's request with a unicast. However, this is only possible if the client already has an IP address assigned (e.g. from a previous session) the lease period (assignment period) of which is not yet expired. Otherwise a DHCP server sends its message to the broadcast address 255.255.255.255. Aside from the available network address (IP address) for the client and the further communication parameters this message also contains the hardware address (MAC address) previously transmitted by the client with the DHCP Discover message. This way the client is able to determine whether the DHCP Offer message of a server is directed to it.

While the client is now waiting for the DHCP Offer message from the servers, a timer is running. If the server doesn't receive a DHCP Offer until the timer has expired, the DHCP Discover message is repeated. If the client has received one or more DHCP Offer messages from one or several servers, it in turn responds with a DHCP Request message as a broadcast. This message contains a server identifier with the designation of the DHCP server the parameters of which the client has chosen. The message is sent as a broadcast to inform all servers about which server the client has decided for. The servers, which were not chosen by the client, determine this fact by comparing the 'server identifier' field with their own designation and then interpret the DHCP request as a rejection. As a result, they release the parameters temporarily reserved for the client which can then be used for requests by other clients.

The server chosen by the client with the DHCP Request then responds with a DHCP Ack message containing all configuration parameters for the client including the assignment period (lease period). The client then tests the parameters (e.g. ARP for the assigned IP address, refer to 0). If the test is successful, the configuration of the client is finished. The client is now able to send and receive TCP/IP packages. If, however, the client detects faulty parameters in the DHCP Ack message, it sends a DHCP Decline message to the server and then repeats the configuration process. The configuration process is also started again if the client receives a DHCP Nak message from a server instead of a DHCP Ack message.

After a successful configuration, the client permanently verifies the expiry of the lease period and, if necessary, sends a DHCP Request message to the configuring server to renew its lease period. If, however, a client wants to finish its address assignment (prematurely), it sends a DHCP Release message to exactly this server.

1.3.8 Address Resolution Protocol (ARP)

On the physical layer diverse mechanisms are used to address the individual devices in the network. For example, Ethernet uses a 48 bit long MAC address. During the installation of TCP/IP protocols a 32 bit long IP address is assigned to this hardware address. Since the physical layer is working completely independent of the above layers, the IP protocol does not know the hardware addresses of the communication parameters. This is why the Address Resolution Protocol (ARP) has been developed as an auxiliary protocol for layer 3. ARP converts IP addresses to the respective network-specific hardware addresses. This address mapping is performed using either the static or the dynamic method.

Static address mapping

For static address mapping, the system administrator has to create tables in the used devices containing the fixed mapping between the hardware and the IP address. In order to setup a connection to a device configured this way, first the respective entry with the IP address of the target node is searched in the table. If a corresponding entry is found, the connection can be established. The use of the static method makes only sense in small networks since each modification of the network or a device requires the tables of all devices to be updated.

Dynamic address mapping

Today, the most common method of converting IP addresses to hardware addresses is the dynamic method. Before data are transmitted via the network, the Internet protocol verifies whether the entry for the target node is present in the ARP address table (ARP cache) of the device. If no corresponding entry can be found, the wanted address is requested from all existing computers in the network using a broadcast (ARP Request). Only those computers respond which have an entry for the wanted IP address. This is at least the wanted computer itself. The response (ARP Reply) is stored in the local ARP cache of the requesting device. The table entry first only exists for 20 minutes before it is deleted by the ARP timer. Each further use of the entry starts the timer again. The local storage of the combination of IP and hardware addresses considerably reduces the access to the network. The time limitation prevents the table size from growing caused by entries which are no longer needed. This way it is furthermore guaranteed that the respective entries are checked and/or updated from time to time. If, for example a computer is replaced by another one with the same IP number but a different hardware address, the ARP entries about the replaced computer in the other devices are invalid until the timer expires.

1.3.9 Other protocols and applications

Aside from the ones described above, numerous further protocols and applications exist in TCP/IP networks. However, since these are not part of the basic functionality and only optionally supported by some devices, they are not described in detail here. In the following sections only some of the best-known representatives of the higher protocols and applications are briefly described.

Hypertext Transfer Protocol (HTTP):

The Hypertext Transfer Protocol (HTTP) is used to exchange hypertext documents between www clients and www servers. Thus, HTTP which is already used as a standard protocol since 1990 is a communication protocol of the application layer. HTTP is based on a connection-oriented transport mechanism. If a client wants to download the data identified by a URL (e.g. www.abb.com/index.html) from a server, it establishes the TCP connection to the server via the logic port 80. The procedure for a communication between a www client (e.g. a PC) and a www server is always as follows:

1. The client establishes a TCP/IP connection to the respective server.
2. After the connection has been established completely, the client transfers the HTTP requests to the server.
3. The server responds to the requests and transfers the corresponding sets of data to the client.
4. Finally the TCP/IP connection is completely closed again.

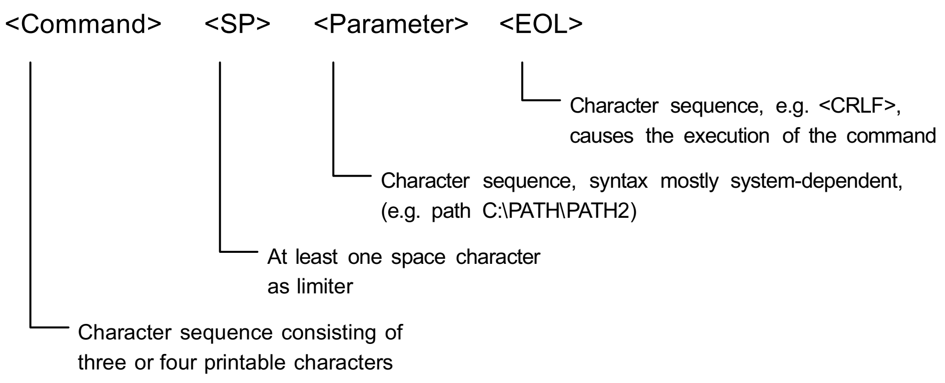
Thus, HTTP is a request/response protocol. The www client transfers the HTTP request to the server. The server reacts with a HTTP response consisting of the response header (status line, used HTTP version, status/error message, server information) and the actual message (message body).

File Transfer Protocol (FTP):

The File Transfer Protocol provides a standardized interface for the exchange of files between computers, independent of their design and operating system as well as for the administration of stored data. FTP is directly based on TCP and described in RFC 959 and the MIL-STD 1780 specifications. The FTP protocol uses port 21 for the control connection and port 20 for the data connection.

Communication between an FTP client (client PI, PI = Protocol Interpreter) and an FTP server (server PI) is performed using a set of specific commands with corresponding acknowledgements. Generally a command is composed of a four-figure character string (e.g. STOR) followed by additional information (e.g. specification of the path). Acknowledgements consist of three-figure character strings with optional text.

The following figure shows the FTP command syntax.



<Command>
<SP>
<Parameter>
<EOL>
Character sequence, e.g. <CRLF>, causes the execution of the command
Character sequence, syntax mostly system-dependent, (e.g. path C:\PATH\PATH2)
At least one space character as limiter
Character sequence consisting of three or four printable characters

Figure 1-15: FTP command syntax

Common FTP implementations have a uniform and partly graphical user interface. So, for example many applications have the commands GET for transferring a file from a remote computer to the local computer and PUT for transferring a file from the local computer to the remote computer. These two commands are not defined by FTP. However, there are existing FTP commands with corresponding functions (GET -> RETR, PUT -> STOR). Consequently it's the application's task to convert user commands to corresponding FTP commands. For the user itself, this conversion is not visible.

Simple Mail Transfer Protocol (SMTP):

The Simple Mail Transfer Protocol enables the transmission of Email messages. SMTP is directly based on TCP and uses the well-known port 25. The protocol is described in RFC 821 and the MIL-STD 1781 standard. SMTP only enables the transmission of Emails via a data network. It is not determined how the message is switched between the user and SMTP nor how the received message is stored and presented to the user. These tasks have to be done by the higher applications.

Messages sent by means of SMTP normally have the format defined in RFC 822. A 822 message is composed of some lines of header information followed by an empty line as a separator and the text. This text is also called the message body. Normally a header line consists of a keyword, a colon and a value (text, address) for the respective keyword. Examples of typical keywords are "from", "to" or "subject". They describe the sender and the recipient of the message as well as its subject. SMTP allows to specify several recipients one behind the other to simultaneously send a message to several recipients. The message body consists of pure text (7 bit ASCII) of any length. The following example shows an SMTP message in 822 format.

```
FROM: Paul Muster <Paul@Muster.COM>
SUBJECT: Confirmation of appointment
TO: Hans.Beispiel@Test.DE

Date on Thursday, 12 a.m. is ok.

Paul
```

Network Information Service (NIS):

The Network Information Service is used for the centralized administration and maintenance of a local network and its users. This is implemented by several databases which are used to administer the computer names, user groups, services and user accounts. NIS furthermore allows to structure a local network and its users into smaller administration units, the so-called domains. Not least because of the missing data encryption and various safety deficiencies, the NIS protocol has later been further developed to NIS+.

Domain Name Service Protocol (DNS):

Prior to the introduction of the Domain Name Service a central host table (hosts.txt) has been administered by the Network Information Center of the Defense Data Networks. The goal of development of the protocol was to replace this table by a distributed database application in order to ease the maintenance of the host and domain names and to reduce and distribute the load caused by the transmission of host tables in the tremendously growing Internet.

DNS is based on UDP (port 53). The domain names used in the Domain Name Service are based on a hierarchical naming concept. A domain name consists of several subnames separated by dots (e.g. sample.example.com). The Domain Name System is structured like a tree, starting from the so-called root. Each node in this tree is a zone (domain). A Domain Name Server always only knows the next higher and the next lower server of the respective domain. A Domain Name Server is responsible for one complete zone. A zone begins in a node and contains all branches included under this node. The highest domain level is called the Top Level Domain. The following figure shows the Top Level Domains for the USA.

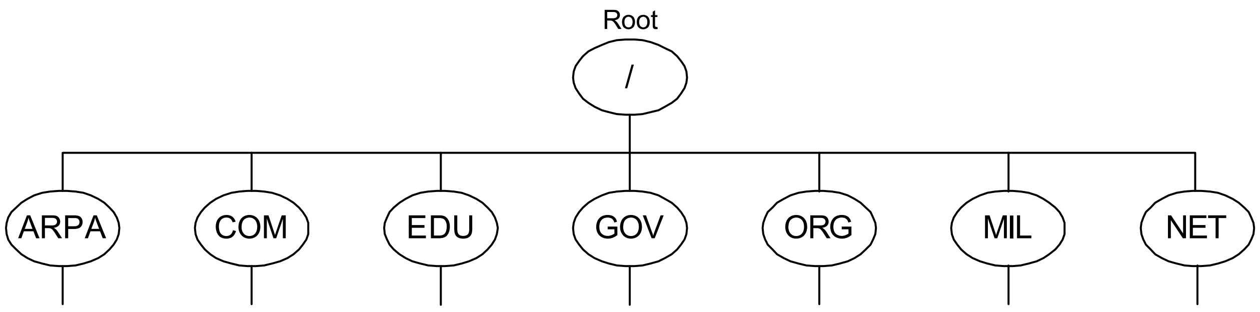


Figure 1-16: Top Level Domains of the USA

The task of a Domain Name Server is to deliver a requesting client computer the IP addresses for a symbolic name.

Internet Name Service (IEN 116):

The Internet Name Service protocol is the oldest name service protocol. It was published 1979 as Internet Engineering Note 116 (IEN 116). This protocol is directly based on UDP and converts logical symbolic names into IP addresses.

If a computer wants to establish a connection to another device in the network but the user only knows the logical device name (e.g. controller1) and not its IP address, the computer first transmits a name request containing the logical device name to the name server. The name server then returns the IP address of the device and the client can now use this IP address to establish the connection to the device. The name server has to be known by the local computer. This is normally specified by the responsible system administrator in a name server file.

Hosts file:

In the previous sections different naming services are described which allow a determination of the IP address of a device by means of its symbolic name. However, if no name server is available in a network or if the device to be called is not registered in the name server, using the host table of the local computer still represents a simple possibility to call the respective device using its symbolic name.

On Windows computers, the hosts file is normally located in the subdirectory ...\etc (e.g. for Windows NT C:\WINNT\SYSTEM32\DRIVERS\ETC\HOSTS). This file is a simple text file containing the assignment of IP addresses to the host names and can be edited using any editor. By default, the hosts file at least contains the loopback address (127.0.0.1 localhost) and, if applicable, the IP address of the local computer. This list can be expanded as desired. For this purpose the entries must have the following structure: IP address of the computer, followed by the respective host name and optional alias names. The IP address and the host name (as well as the alias names) have to be separated by at least one blank character. The # character identifies the beginning of a comment. The end of a comment is automatically defined by the line end. Comments are ignored by the system when interpreting the hosts file.

The following example shows a hosts file.

```
#Loopback address for localhost
127.0.0.1 localhost

#System part 1
193.0.4.1 Head office
193.0.4.2 Visualization #Control Room
193.0.4.3 Controller1: Silo
193.0.4.4 Controller2 Pumping station

#System part 2
193.0.5.1 Mixer
193.0.5.2 Heating
193.0.5.3 Control cabinet
```

1.4 Cabling

1.4.1 Network cables

There are numerous standards regarding the Ethernet cabling which are to be observed when configuring a network. So, the standards TIA/EIA-586-A and ISO/IEC 11801 define the properties of the cables to be used and divide the cables into categories. While the TIA/EIA specification is more directed to the American market, the ISO/IEC standard meets the international requirements. Furthermore, the European standard EN 50173 which is derived from ISO/IEC 11801 represents the standard to be observed for structured cabling.

Usually twisted-pair cables (TP cables) are used as transmission medium for 10 Mbit/s Ethernet (10Base-T) as well as for 100 Mbit/s (Fast) Ethernet (100Base-TX). For a transmission rate of 10 Mbit/s, cables of at least category 3 (IEA/TIA 568-A-5 Cat3) or class C (according to European standards) are allowed. For Fast Ethernet with a transmission rate of 100 Mbit/s, cables of category 5 (Cat5) or class D or higher have to be used. The following table shows the specified properties of the respective cable types per 100 m.

Parameter	**10Base-T [10 MHz]**	**100Base-TX [100 MHz]**
Attenuation [dB / 100m]	10.7	23.2
NEXT [dB / 100m]	23	24
ACR [dB / 100m]	N/A	4
Return loss [dB / 100m]	18	10
Wave impedance [Ohms]	100	100
Category	3 or higher	5
Class	C or higher	D or higher

Table 1-6: Specified cable properties

The TP cable has eight wires where always two wires are twisted to one pair of wires. Different color codes exist for the coding of the wires where the coding according to EIA/TIA 568, version 1, is the most commonly used. In this code the individual pairs are coded with blue, orange, green and brown color. While one wire of a pair is single-colored, the corresponding second wire is colored alternating with white and the respective color. For shielded cables it is distinguished between cables that have one common shielding around all pairs of wires and cables that have an additional shielding for each pair of wires. The following table shows the different color coding systems for TP cables:

Pairs	**EIA/TIA 568 Version 1**		**EIA/TIA 568 Version 2**		**DIN 47100**		**IEC 189.2**	
Pair 1	white/blue	blue	green	red	white	brown	white	blue
Pair 2	white/orange	orange	black	yellow	green	yellow	white	orange
Pair 3	white/green	green	blue	orange	grey	pink	white	green
Pair 4	white/brown	brown	brown	slate	blue	red	white	brown

Table 1-7: Color coding of TP cables

Two general variants are distinguished for the pin assignment of the normally used RJ45 connectors: EIA/TIA 568 version A and version B where the wiring according to EIA/TIA 568 version B is the most commonly used variant.

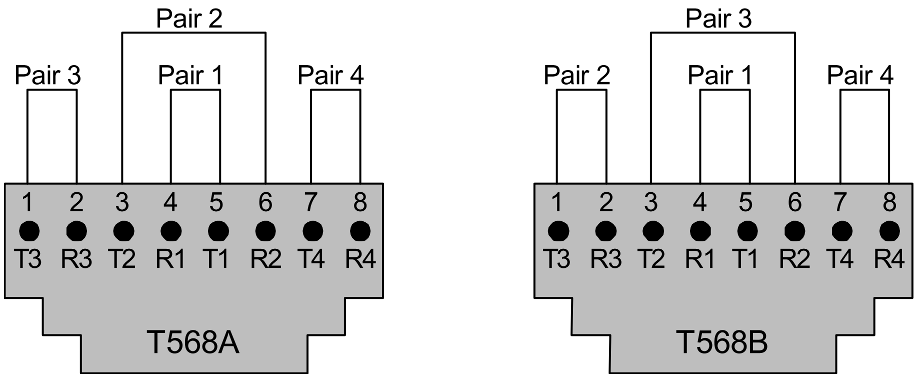


Figure 1-17: Pin assignment of RJ45 sockets

1.4.2 Connector pin assignment

Today, RJ45 connectors are accepted as a standard for the cabling of 10 Mbit/s networks (10Base-T) as well as for 100 Mbit/s networks (100Base-TX). Generally, the same pin assignment is used for both variants.

Pin number	Signal
1	TD+ (data output)
2	TD- (data output)
3	RD+ (data input)
4	-
5	-
6	RD- (data input)
7	-
8	-

Table 1-8: Pin assignment of RJ45 connectors

1.4.3 1:1 cables and crossover cables

A direct connection of two terminal devices is the simplest variant of an Ethernet network. In this case, a so-called crossover cable (also called crossconnect or crosslink cable) has to be used to connect the transmission lines of the first station to the reception lines of the second station. The following figure shows the wiring of a crossover cable.

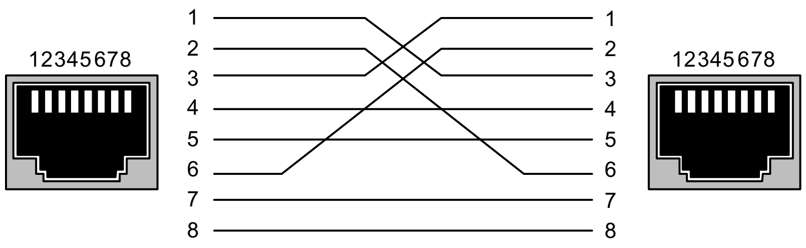


Figure 1-18: Wiring of a crossover cable

For networks with more than two subscribers, hubs or switches have to be used additionally for distribution (see also section "1.4 Cabling"). These active devices already have the crossover functionality implemented which allows a direct connection of the terminal devices using 1:1 cables.

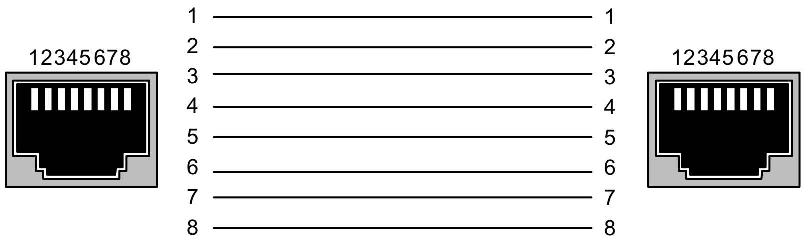


Figure 1-19: Wiring of a 1:1 cable

1.4.4 Cable length restrictions

For the maximum possible cable lengths within an Ethernet network various factors have to be taken into account. So, for twisted pair cables (for transmission rates of 10 Mbit/s and 100 Mbit/s) the maximum length of a segment which is the maximum distance between two network components is restricted to 100 m due to the electric properties of the cable.

Furthermore, the length restriction for one collision domain has to be observed. A collision domain is the area within a network which can be affected by a possibly occurring collision (i.e. the area the collision can propagate over). This, however, only applies if the components operate in half duplex mode since the CSMA/CD access method is only used in this mode. If the components operate in full duplex mode, no collisions can occur. Reliable operation of the collision detection method is important which means that it has to be able to detect possible collisions even for the smallest possible frame size of 64 bytes (512 bits). But this is only guaranteed if the first bit of the frame arrives at the most distant subscriber within the collision domain before the last bit has left the transmitting station. Furthermore, the collision must be able to propagate to both directions within the same time. Therefore, the maximum distance between two ends must not be longer than the distance corresponding to the half signal propagation time of 512 bits. Thus, the resulting maximum possible length of the collision domain is 2000 m for a transmission rate of 10 Mbit/s and 200 m for 100 Mbit/s. In addition, the bit delay times caused by the passed network components have also to be considered.

1.5 Network components

The topology of an Ethernet network is like a star or tree structure. Up to two stations can be connected to each segment where active distribution devices like hubs or switches are also considered as a station. The following figure shows an example of a simple Ethernet network.

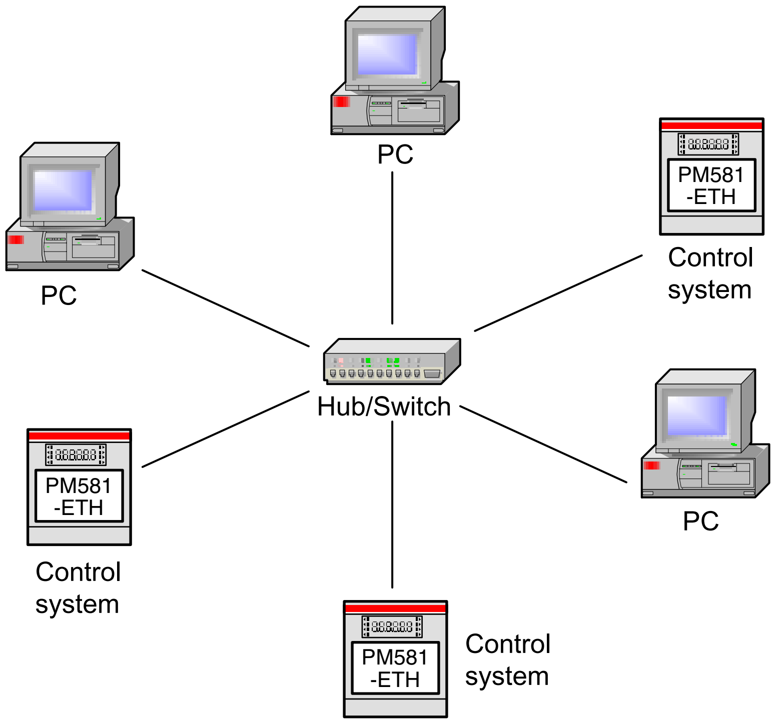


Figure 1-20: Example of a simple network

The following sections introduce the different types of components required for a network.

1.5.1 Terminal devices

Terminal devices are devices that are able to send and receive data via Ethernet, e.g. controllers with an Ethernet coupler or PCs with an integrated network adapter. With this, one of the essential functions of a network adapter is to transfer all data packages to the PC itself instantly and without any loss. Occurring defiles or even errors can cause data packages to be lost. Such losses of data have to be got under control by higher protocols (e.g. TCP/IP) which results in considerable performance reductions. The direct implementation of higher protocols on the network adapter can increase the performance and save the resources of the host system (e.g. controller).

1.5.2 Repeaters and hubs

At the dawning of the Ethernet, the repeaters had only two network connections and were used to connect two segments to each other in order to extend the segment length. Later, repeaters with more than two network connections were available. Those star distributors are called hubs. They are able to connect several segments. Apart from the number of network connections the functionality of hubs and repeaters is identical. This is why we only use the term "hub" in the following descriptions.

Hubs are operating on the lowest layer of the ISO/OSI model and are therefore independent of the protocols used on Ethernet. The network connections of hubs are exclusively operated in half duplex mode. Due to this, collision domains can freely propagate beyond the hubs. A hub can only support one transmission rate for all connections. Therefore it is not possible to connect segments with different transmission rates via a simple hub. For this purpose a dual-speed hub has to be used. The fundamental functions of hubs are as follows:

- Restoration of the signal magnitude
- Regeneration of the signal timing
- Propagation of a detected collision
- Expansion of short fragments
- Creation of a new preamble
- Isolation of a faulty segment

When transmitted over the medium (e.g. a twisted pair cable) the data signal is attenuated. The task of a hub is to amplify an incoming signal in order to make the full signal magnitude available at the outputs again. Furthermore, a distortion of the binary signal's on-off ratio (jitter) can occur during data transmission. When transmitted via a hub, the hub is able to restore the correct on-off ratio of the signal which avoids propagation of the signal jitter beyond the segment.

However, one of the most important tasks of a hub is to propagate occurring collisions within the entire collision domain so that the collision can be detected by all connected stations. If it detects a collision on one of his connections, the hub sends a so-called jam signal over all connections. If a hub receives a data fragment which, by its principle, could only be created by a collision, it first brings the fragment to a length of 96 bits and then forwards it via the ports. This shall guarantee that the data fragment can be received by all stations independent of their distance to the hub and removed from the network. The detected data fragments are removed by the terminal devices by not forwarding them to the higher layers.

By means of the data package preamble the beginning of a data package is detected so that the recipient can synchronize to the incoming data stream. However, during the data transmission it can occur that the first bits of a preamble are lost. The task of the hub is to restore a possibly incomplete preamble before forwarding it.

If collisions occur within one segment in large numbers in a short period of time or if e.g. a short circuit on a data line causes failures, the hub switches off the faulty segment to avoid interference to the entire collision domain.

10 Mbit/s hubs:

The 10Base-T connections of a 10 Mbit/s hub are implemented as MDI-X ports and therefore already crossed internally. The advantage is that the terminal devices can be directly connected using 1:1 twisted pair cables and no crossover cables are required. Some hubs additionally have a so-called uplink port which can be used to connect another hub. In order to also enable the use of a 1:1 cable for this port, it is implemented as a normal non-crossed MDI port. In many cases this port can also be switched between MDI and MDI-X or is implemented as a double port with two connections in parallel (1 x MDI, 1 x MDI-X). In this case, it has to be observed that these parallel ports may only be used alternatively and not at the same time.

Hubs are normally equipped with several LEDs for status indication. So, for example a Link LED indicates the correct connection between the terminal device and port at the hub. This way, incorrect cabling can be quickly detected. Further LEDs indicate for example the data traffic over a port or the collisions.

The maximum permitted number of 10 Mbit/s hubs within one collision domain is limited to 4. This restriction is due to two reasons. One reason is that the bit period time delay, which is inevitably increased by each hub, must not exceed 576 bit periods. The second reason is that the so-called interframe gap (IFG) must not be shorter than at least 47 bit periods. The interframe gap describes the time interval between two data packages and shall allow the receiving stations to recover from the incoming data stream. However, the regeneration of an incomplete preamble performed by the hub reduces the time between the data packages due to the completion of possibly missing bits.

One possibility to get round the restriction to four hubs is the use of stackable hubs. These hubs are connected to each other via a special interface instead of using the uplink port and therefore constitute one logic unit. As a result, they appear as one single big hub to the external.

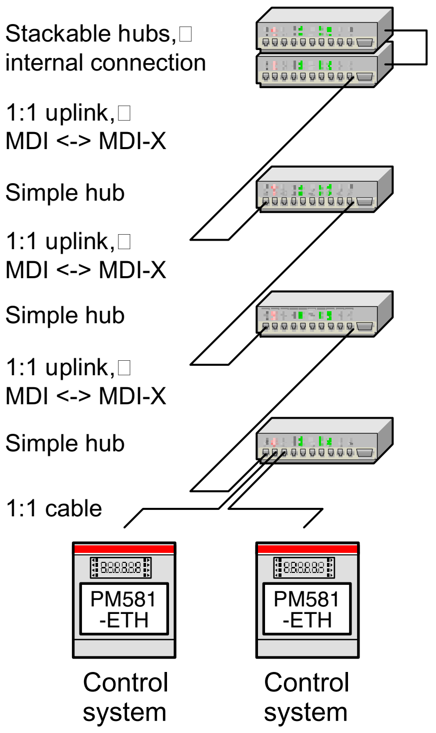


Figure 1-21: Stackable hubs

100 Mbit/s hubs:

The principle operation of 100 Mbit/s hubs is like the 10 Mbit/s hubs. However, the hubs for 100 Mbit/s Ethernet have to be additionally distinguished to class I and class II hubs.

Class I hubs (or class I repeaters) are able to connect two segments with different transmission media. For this purpose the complete data stream has to be decoded on the receiving side and encoded again on the transmission side according to the transmission medium. This conversion process leads to higher delay times. Due to this, only one class I hub is permitted within one collision domain.

In contrast, class II hubs support only one transmission medium. No conversion of the data stream is required. This leads to shorter delay times compared with class I hubs. This is why for two segments with a maximum length of 100 m each, up to two class II hubs which are again connected to each other via a 5 m long segment can be used within one collision domain.

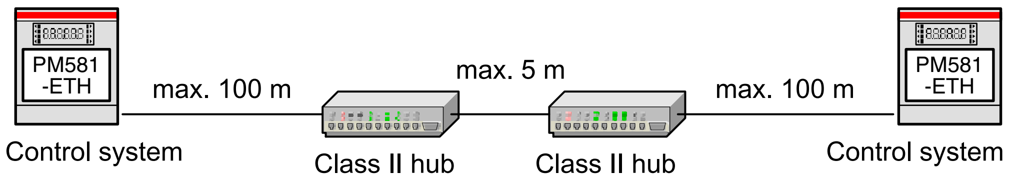


Figure 1-22: Use of a class II hub

10/100 Mbit/s dual-speed hubs

In contrast to the simple hubs, dual-speed hubs are able to support two transmission rates and thus enable to connect two Ethernet networks with different data rates to each other. Dual-speed hubs are internally structured like two separate hubs or paths (one for each data rate). By means of the auto negotiation function the transmission rate of the connected station is determined and automatically switched to the corresponding path. Each internal path is a separate hub. For the temporary storage of the data packages, the paths are connected to each other via an internal switch. Dual-speed hubs likewise operate in half duplex mode. However, the internal switch provides a clear separation of the 10 Mbit/s and the 100 Mbit/s side so that unlike the simple hubs a collision domain cannot reach beyond the borders of the corresponding side of the dual speed hub.

1.5.3 Bridges, switches and switching hubs

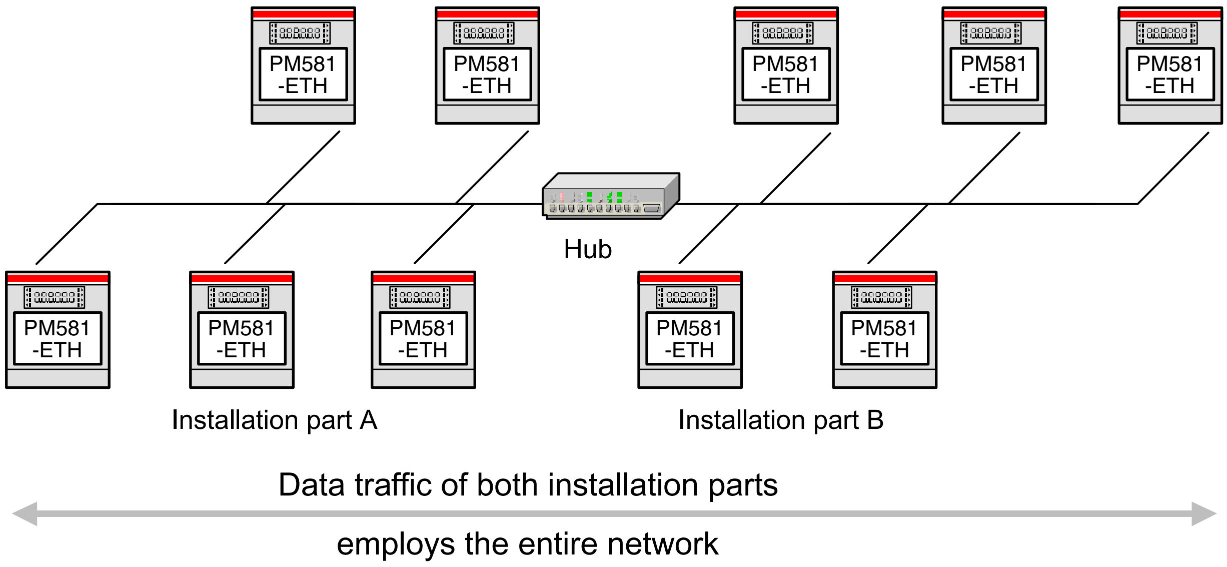
Basically the terms bridge, switch and switching hub designate the same. In the early beginning of the Ethernet, the term bridge was formed by the fact that a bridge had only two network connections. Later, so-called multiport bridges with several connections came up which were also called switches or switching hubs. This is why we use the common term "switch" in the following descriptions for all the components mentioned above.

The use of a switch is another variant of connecting network segments to each other. The decisive difference between a hub and a switch is that a switch is operating on the second layer of the ISO/OSI model, the MAC layer.

The following sections describe the functionality of such a layer 2 switch. For reasons of completeness it has to be mentioned that switches operating on higher and therefore protocol-specific layers also exist.

Using a switch, load separation between networks can be implemented which leads to an increased performance due to the reduced load of the individual segments. In contrast to a hub, a switch does not operate transparently (i.e. it doesn't forward all data packages via all ports) but decides on the basis of the MAC target address whether and via which port an incoming data package has to be forwarded. The data package is only forwarded if the target station is located in another segment or if the target address of the data package contains a multicast or broadcast address.

As already mentioned, the decisive advantage of a switch is the logical separation of networks. Therefore, a switch represents a border for a collision domain. Aside from the performance improvement, the use of a switch allows a network to be extended beyond the usual borders.


PM581
-ETH
Hub
Installation part A
Installation part B
Data traffic of both installation parts
employs the entire network

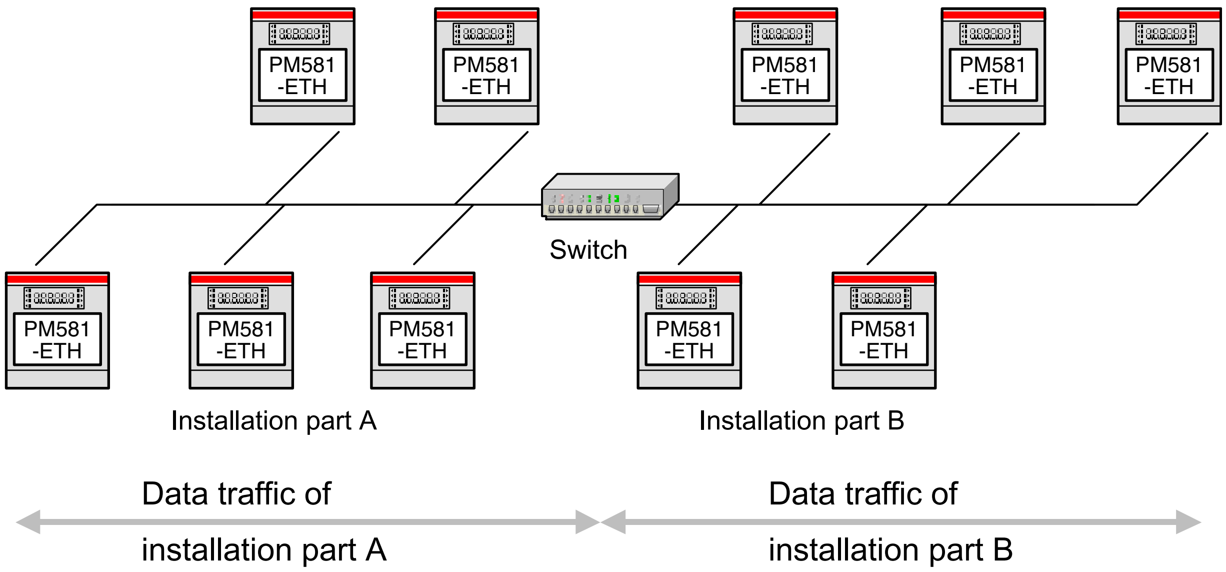


Figure 1-23: Use of hubs and switches

To enable crosswise traffic between the segments, a switch has to be able to temporarily store the incoming data packages until they can be transmitted on the forwarding segment. The decision about forwarding of data packages is done using address tables. These address tables are generated by the switch itself during a self-learning process. During this process, the switch remembers the source addresses (MAC addresses) of incoming data packages of a port. If it later receives further data packages, the switch compares their target addresses with the entries in the address tables of the ports and, in case of a match, forwards the respective package via the corresponding port. Here, the following cases have to be distinguished:

- If the source station and the target station are located within the same segment, the data package is not forwarded.
- If the station of the target address is located in another segment than the source station, the data package is forwarded to the target segment.
- Data packages containing a multicast or a broadcast address as the target address are forwarded via all ports.
- A data package with a target address which is not contained in the address tables is forwarded via all ports (Frame Flooding).

The latter case normally only occurs during the first time after starting a switch since the address is usually entered after some time when exchanging a data package.

In order to limit the size of the address tables, addresses which are not used over a longer period of time are additionally removed from the tables. This also avoids incorrect forwarding as it would appear e.g. when a station is moved within the network.

To enable the building of a redundant network structure (as it is often found in more complex networks) using switches, the so-called spanning tree method has been introduced. With this method, the switches exchange configuration messages among themselves. This way the optimum route for forwarding data packages is determined and the creation of endless loops is avoided. The exchange of messages is performed cyclic. As a result a connection breakdown is detected and forwarding is automatically changed to another route.

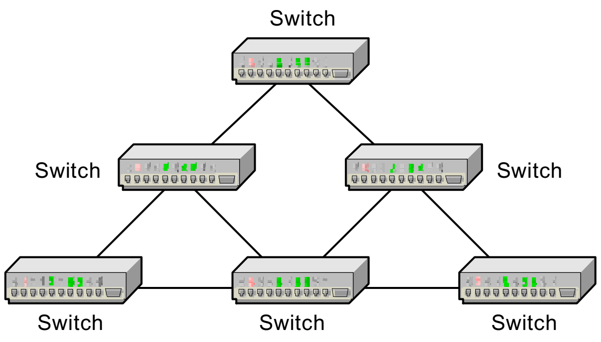


Figure 1-24: Redundant network structure using switches

Using a switch instead of a hub increases the bandwidth of the individual segments and therefore leads to an increased performance. Building a network consistently with switches furthermore enables full duplex operation and thus simultaneous data traffic in both directions since switches are able to establish dedicated peer-to-peer connections between the individual ports. The use of the access method CSMA/CD is not required since collisions can no longer occur. Depending on the network structure, this can further increase the performance drastically. For full duplex connections furthermore no length restrictions of the collision domain have to be observed.

1.5.4 Media converters

Media converters provide the possibility of connecting components to each other via different media. The most frequently occurring case for this is the conversion between twisted pair (TP) and fibre optic cabling.

When using media converters it has to be observed that a connected port operating in half duplex mode is no collision domain border. This is often not considered when using optical fibres to bridge a larger distance. The fibre optical port of a media converter furthermore does not support the auto negotiation function. Due to this, if an Ethernet component is directly connected to the fibre optic side of a media converter, the transmission mode has to be set fixed according to the component connected on the twisted pair side. If a connection between two twisted pair components is established using two media converters, it is absolutely required that both twisted pair components are operating with the same transmission mode. If necessary, manual setting has to be performed.

1.5.5 Routers

Routers connect networks with identical protocols or addressing mechanisms. The main task of a router is to perform the routing for the transmission of data packages from the sender to the recipient. Routers are able to effectively reduce the data traffic between individual networks by using different algorithms. The dynamic routing leads to a load reduction for the entire network. If the router has several alternative routes to the target station available, it will always choose the optimum way depending on the current load on the network and the expected costs.

In contrast to the switches which forward the packets on the basis of layer 2 (e.g. Ethernet), the routers operate on layer 3 (e.g. IP). While the switches forward the packets on the basis of the MAC addresses, the routers evaluate the contained IP addresses. For this purpose, when receiving a data package a router first has to remove the outer telegram frame in order to be able to interpret the addresses of the inner protocol and then it has to re-assemble the data package again before forwarding it. This results in higher latency periods (time of stay) of the data within the router itself. The investigation of a data package necessary for routing makes clear that a router has to be able to process all network protocols to be routed over this router. Due to the increasing spread of heterogeneous networks, today often routers are used which are able to support several network protocols (e.g. IP, IPX, DECnet, AppleTalk) instead of special IP routers. Such routers which are able to process several network protocols are called multi-protocol routers. Some routers additionally have a bridge functionality (bridge routers, Brouters) which enables them to also forward the data packages of protocols a router cannot interpret or which do not support the routing function (e.g. NetBios).

1.5.6 Gateways

A gateway is a computer which is able to couple completely different networks. Gateways are operating on a layer above layer 3 of the ISO/OSI model. They are used to convert different protocols to each other. For the connected subnetworks, a gateway is a directly addressable computer (node) with the following tasks:

- Address and format transformation
- Conversions
- Flow control
- Necessary adaptations of transmission rates for the transition to the other subnetworks.

Gateways can furthermore be used to implement safety functions on the application layer (firewalls). For example, gateways are used for the coupling of PCs located in local area networks (LAN) to public long distance communications (wide area networks, WAN).

2 The Ethernet coupler

2.1 Features

2.1.1 Supported protocols

IP - Internet Protocol (RFC 791):

- Freely configurable IP address and network mask
- Configurable IP address of the standard gateway
- IP datagram size: 1500 bytes max.
- Route Cache size: 32 entries
- Route Timeout: 900 seconds
- Number of IP multicast groups: 64 for reception, unlimited for transmission

TCP - Transmission Control Protocol (RFC 793, RFC 896):

- Amount of user data for TCP telegrams: 1460 bytes max.

UDP - User Datagram Protocol (RFC 768):

- Amount of user data for UDP telegrams: 1472 bytes max.

BOOTP - Bootstrap Protocol (RFC 951, RFC 1542, RFC 2132):

DHCP - Dynamic Host Configuration Protocol (RFC 2131, RFC 2132):

OpenModbus:

- Client and/or server mode (several times)
- Up to 8 simultaneous client or server connections
- Supported function codes: 1, 2, 3, 4, 5, 6, 7, 15, 16
- Maximum amount of data per telegram: 100 coils (words) or 255 registers (bits)
- Configurable connection monitoring functions

NetIdent:

- Devices can be identified and accessed via the network (even unconfigured devices)
- Unique identification and localization via rotary switch on the devices

ARP - Address Resolution Protocol (RFC 826):

- ARP Cache size: 64 entries
- ARP Timeout: 600 seconds

ICMP - Internet Control Message Protocol (RFC 792):

IGMPv2 - Internet Group Management Protocol, version 2 (RFC 2236):

Further protocols and applications are in preparation

2.1.2 Sockets

- Number of sockets: 16
- Socket options can be set individually

2.1.3 Restrictions

- IP fragmentation is not supported
- TCP Urgent Data is not supported
- TCP port 0 is not supported
- TCP port 502 is reserved for OpenModbus
- TCP port 1200 is reserved for gateway access
- UDP port 67 is reserved for BOOTP and DHCP
- UDP port 25383 is reserved for NetIdent protocol
- UDP port 32768 is reserved for UDP blocks

2.2 Technical data

2.2.1 Technical data of the coupler

Internal Ethernet coupler PM5x1-ETH

Coupler type	Ethernet coupler, internal
Processor	EC1-160, 48 MHz
Ethernet controller	EC1-160 internal
Internal power supply with	+5 V, 280 mA typ.
Internal RAM memory (EC1)	256 kbytes
External RAM memory	-
External Flash memory	512 kbytes (firmware)
CE sign	yes

External Ethernet coupler CM577

Coupler type	Ethernet coupler, external
Processor	EC1-160, 48 MHz
Ethernet controller	EC1-160 internal
Internal power supply with	+5 V, 420 mA typ.
Internal RAM memory (EC1)	256 kbytes
External RAM memory	2 x 128 kbytes (web server)
External Flash memory	512 kbytes (firmware), 2 Mbytes (web sever)
CE sign	yes

2.2.2 Interfaces

Internal Ethernet coupler PM5x1-ETH

Ethernet	10/100 Base-TX, RJ45 socket
LED indication	none
Station identification	input via keypad, 0...255

External Ethernet coupler CM577

Ethernet	10/100 Base-TX, internal switch, 2 x RJ45 socket
LED indication	status indication via 5 LEDs
Station identification	rotary switch, 0...255 (00...FF$_{hex}$)

2.2.3 Technical data of the Ethernet interface

Transmission mode	Half or full duplex operation, adjustable
Transmission rate	10 or 100 Mbit/s, adjustable
Auto negotiation	optionally adjustable
MAC address	optionally configurable
Ethernet frame types	Ethernet II (RFC 894), IEEE 802.3 receive only (RFC 1042)

2.3 Connection and data transfer media

2.3.1 Attachment plug for Ethernet cable

8-pole RJ45 plug

See also chapter "1.4 Cabling".

Assignment:

Pin No.	Signal	Meaning
1	TxD+	Transmit data (line) +
2	TxD-	Transmit data (line) -
3	RxD+	Receive data (line) +
4	NC	Not used
5	NC	Not used
6	RxD-	Receive data (line) -
7	NC	Not used
8	NC	Not used
Shield	Cable shield	

Table 2-1: Pin assignment of the attachment plug for the Ethernet cable

2.3.2 Ethernet cable

For structured Ethernet cabling only use cables according to TIA/EIA-586-A, ISO/IEC 11801 or EN 50173 (see also chapter "1.4 Cabling").

2.4 Ethernet implementation

2.4.1 Configuration

The Ethernet coupler is configured via PC using the SYCON.net configuration software . The configuration data created with SYCON.net have to be downloaded to the controller separately (in addition to a CoDeSys project). In the controller, the data are automatically stored in the Flash memory.

If the user defined project is written to SMC, the configuration data are also saved automatically.

2.4.2 Running operation

The integrated protocols are automatically processed by the coupler and the operating system of the controller. The coupler is only completely ready for operation if it has been configured correctly before. Online access via the Ethernet coupler is available at any time, independent of the controller's state. Designable protocols (e.g. fast data exchange via UDP) require function blocks. They are only active while the user program is running. If the user program is stopped, all the planned connections possibly still existing are closed automatically.

2.4.3 Error diagnosis

The Ethernet coupler's operating condition as well as possible communication errors are indicated by LEDs (in case of external couplers) or on the display (in case of internal couplers). Malfunctions of the integrated protocol drivers or the coupler itself are indicated via the Ex error flags and the corresponding LEDs . In case of designable protocols, information about occurring errors are additionally available at the outputs of the corresponding function blocks.

2.5 Diagnosis

Status LEDs of the external Ethernet coupler:

Status indication via 5 LEDs:

LED	Color	Meaning
PWR	green	Supply voltage
RDY	yellow	The coupler is ready for operation.
RUN	green	Status of configuration and communication.
STA	yellow	Status of Ethernet communication.
ERR	red	Communication error

Table 2-2: Status LEDs

Ethernet error messages:

The Ethernet error messages are listed in the following topic: "Error tables of the Ethernet coupler ".

Function blocks:

In case of designable protocols (e.g. fast data exchange via UDP), information about possibly occurring errors are additionally available at the outputs of the corresponding function blocks.

Online diagnosis:

The field bus configuration tool SYCON.net provides extensive online diagnosis functions (refer to documentation for the field bus configuration tool SYCON.net.

3 Designing and planning a network

3.1 Introduction

To obtain optimum performance within a network, it is absolutely essential to plan the network beforehand. This applies to both the initial installation as well as its expansion. Rashly installed networks can not only cause poor network performance, they even can lead to a loss of data since restrictions given by the standard are possibly not kept. At first glance, designing a network causes additional costs, but it will later reduce maintenance expenditures during operation.

The following sections shall explain some principle methods for determining a suitable network structure and give some hints how to find out the network utilization and performance.

3.2 Concepts for structuring a network

Three fundamental aims are to be considered when designing the concept of a network: Performance, quality and safety. The performance of a network is primarily described by the data throughput (as high as possible) and the transmission delay (as short as possible). Quality means stability, fail-safety and availability of the network. The safety aspect considers the safety of the transmitted data, i.e. protection against access to confidential data by unauthorized persons. Whereas performance and quality are planning goals for all kinds of networks, safety has mainly to be considered for networks which can be accessed from the "outside world". For example, a "closed" network inside an installation containing only automation components does not require particular protection of the data against unauthorized access.

Therefore, a detailed requirements analysis has to be done prior to the actual conception of a network in order to meet the specific requirements of the particular network.

Apart from planning the passive structured cabling, making a network conception also includes the selection of suitable active components such as hubs, switches or routers. Planning a new network starts with a registration of all systems (e.g. controllers) to be installed which shall be connected by the network and the requirements to the intended data exchange between these systems. When expanding or optimizing an existing network, first the actual situation has to be determined and the performance of the existing components with regard to the new requirements has to be assessed.

Regarding the network technology the following three general models are distinguished:

- Hierarchy model
- Redundant model
- Safe model

The selection of the suitable model as a basis for planning a network depends on the specific requirements of the installation. Office networks are typically built up based on the hierarchy model since the individual clients do not very often exchange data with each other but only periodically contact the server. Installation-internal networks which do not have any connection to the company network often only consist of automation devices and do not have a server. The connected controllers transmit data in short intervals directly to each other. Furthermore, the operational safety of installation-internal networks has a higher importance since data transmission malfunctions can result in incorrect behavior of the installation or even in production stops. In such cases it is more suitable to choose the redundant model or a safe model.

In the end, all three models shown above are based on the use of switching hubs (switches). Whereas in the past simple hubs were increasingly used to set up a network wherever permitted by the requirements, today almost exclusively switches are used. Using switches, historic Ethernet rules such as the length restrictions of a collision domain no longer have to be observed. This considerably simplifies the network design. Even though the use of switches could make us believe that networks can be expanded to infinite size, it has to be considered that each switch involved in a data transfer causes a delay. Therefore, the IEEE-802.1d bridging standard recommends to limit the number of switches to be passed between two terminal devices to a maximum of seven switches.

3.2.1 Hierarchy model

The hierarchy model intends the subdivision of the network into several levels and a graduation of the data rate between the individual levels. For this purpose, normally at least two grades are used e.g. by connecting the server with a data rate of 100 Mbit/s to the network and the clients with 10 Mbit/s. The advantage of this design is that the server has 10 times the bandwidth of the clients available which enables the server to provide sufficient bandwidth and response time for several clients. Despite the fact that 10 times the bandwidth does not mean that 10 clients can simultaneously access the server, the data transmitted to or from the clients do only need one tenth of the time. In total, this reduces the response time for each individual client.

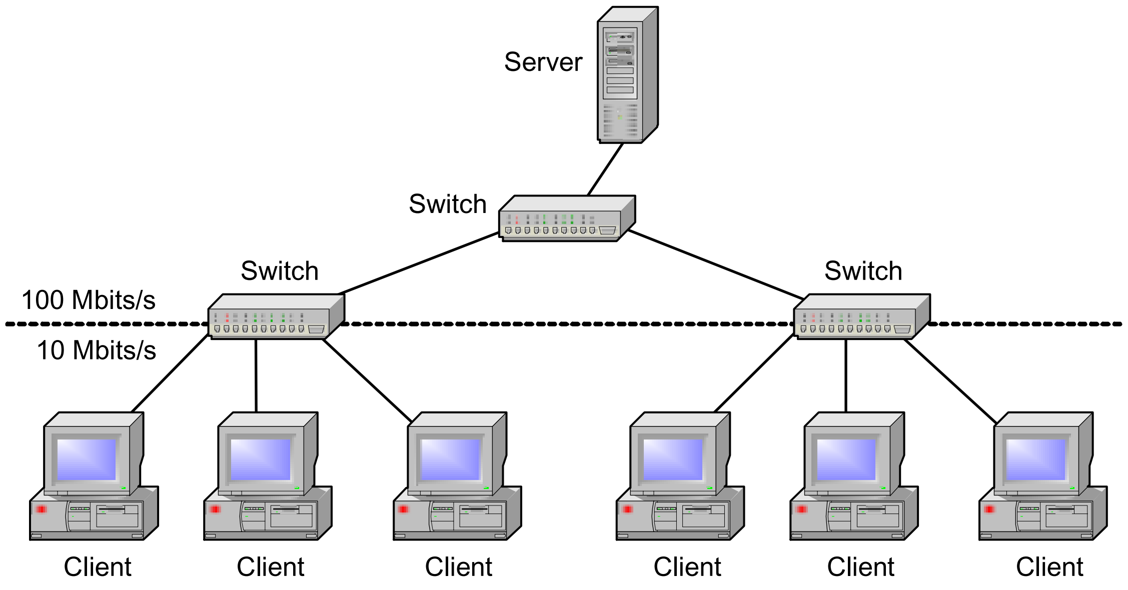


Figure 3-1: Hierarchy model

When dimensioning the individual levels the utilization of the particular level has to be considered. Devices connected to each other via hubs can only be operated in half-duplex mode. Consequently they have to share the commonly used network (shared media). If the utilization of such a shared media is higher than 40 % over a longer period of time, a switch should be used instead of a hub in order to subdivide the collision domain and thus remove load from it. The utilization threshold within such a switched media is 80 %. If this value is exceeded, the utilization should be reduced by selecting a smaller grouping.

3.2.2 Redundant model

The meshed Ethernet structure is a typical example for a redundant network model. To obtain fault tolerance, several connections have to be established between switches or nodes. This way, data exchange can be performed using another (redundant) connection if one connection fails.

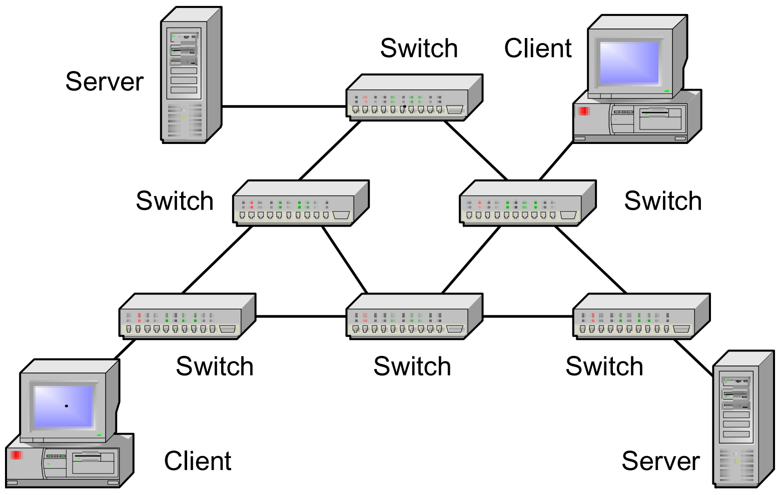


Figure 3-2: Redundant model

However, this meshed constellation leads to loops which would make well-ordered data exchange impossible. The loops would cause the broadcast or multicast data packages to endless stray in the network. In order to suppress such loops, the spanning tree mechanism (refer to 1.5.3 Bridges, switches and switching hubs) is used which always activates only one unique connection and deactivates all other possible connections. On the occurrence of a fault (e.g. caused by an interruption of the network line) the redundant connection is re-activated and then maintains communication between the switches. However, switching of the connection is not without interruption. The time needed for switching depends on the size and structure of the network.

The use of link aggregation which is often also called "trunking" likewise provides increased transmission reliability. Link aggregation actually means the parallel connection of several data lines. This way the bandwidths of the individual data lines are bundled in order to increase the total bandwidth. Furthermore, the parallel connection establishes a redundant connection. If one data line fails, the data can still be transmitted via the remaining lines even though only with reduced bandwidth.

3.2.3 Safe models

To obtain a certain grade of safety for the transmitted data against unauthorized access or to optimize the network utilization, it is suitable to design so-called Virtual Bridged Local Area Networks (VLANs). In a VLAN the data flow is grouped. The simplest variant of a VLAN is obtained by a port-related grouping which means that particular ports of a switch are assigned to a VLAN and data exchange is then only performed within this VLAN. A VLAN can be considered as a group of terminal stations which communicate like in a usual LAN although they can be located in different physical segments. In the end, establishing VLANs leads to a limitation of the broadcast domains. As a result, all subscribers of a VLAN only receive data packages which have been sent by subscribers of the same VLAN. Independent of their physical location, all subscribers of a VLAN are logically put together to one broadcast domain. The limitation of the broadcast domains relieves load from the network and provides safety since only the members of the VLAN are able to receive the data packages.

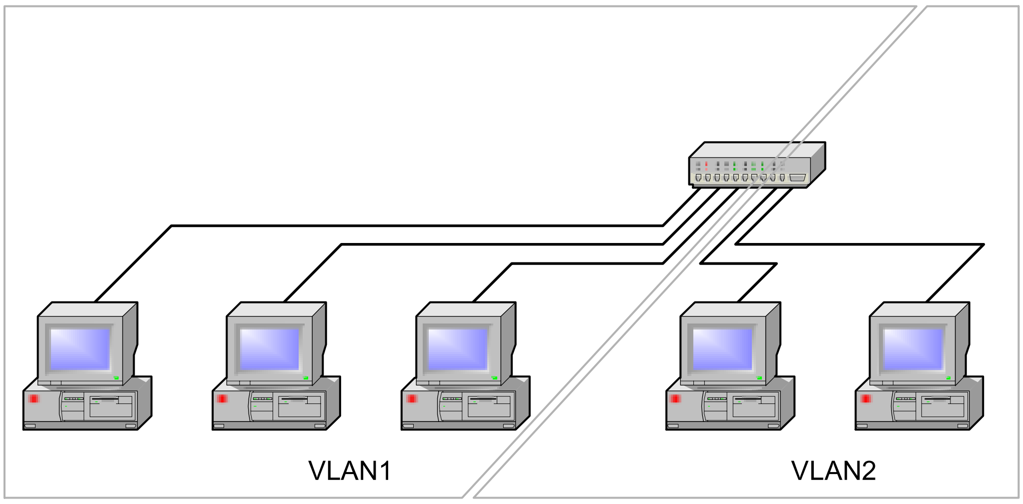


Figure 3-3: Safe models

In order to enable a terminal device connected to a switch to exchange data beyond the borders of the VLAN, the port of the switch has to be assigned to several VLANs. Apart from the simple variant of the port-based VLAN, it is also possible to establish VLANs by evaluating additional information contained in the Ethernet frames.

3.3 Utilization and performance

In the description of the network models it has already been mentioned that the existing hubs should be replaced by switches in order to subdivide the collision domain and thus remove load, if the utilization of a shared media is higher than 40 % over a longer period of time. If the utilization within such a switched media is permanently above 80 %, it is recommended to further relieve load by performing smaller grouping.

However, a network should basically not be dimensioned for the burst utilization. During normal operation usually many smaller data packages are transmitted rather than large data streams. This means that the network load regarding the bandwidth is not as high. Nevertheless, if any bottle-necks occur, the simplest method to eliminate them is to increase the data rate (e.g. from 10 Mbit/s to 100 Mbit/s). In existing networks, however, this is not always possible without problems since the cable infrastructure is possibly not suitable for the higher data rate and the expenditure for a new cabling is possibly not defensible. The only solution in such cases is a segmentation of the network which results in a reduction of the number of devices within the network or collision domain and thus provides more bandwidth for the remaining devices.

A segmentation of a network can be obtained with routers, bridges or switches. However, segmentation is only meaningful if the 80/20 rule is considered and observed. The 80/20 rule says that 80 % of the data traffic have to take place within the segment and only 20 % of the data traffic are forwarded to another segment. This is why a previous analysis of the network traffic is required to enable meaningful grouping. In this analysis it has to be determined which station is communicating with which other stations in the network and which amount of data is flowing for this communication. For shared media the network should be divided in a way that stations producing roughly the same load should be grouped in one collision domain, if it is not possible to make a division based on the communication paths. This way it is guaranteed that stations with lower data traffic are able to meet the typical requirements regarding short response times. Stations with permanently high data traffic generally cause a drastic increase of the response times.

Best performance increase can be obtained by using switches and connecting each single station directly to the switches. This way each station has its own connection to a switch and thus can use the full bandwidth of a port in full-duplex mode. This subdivision and the provision of the dedicated connections is called micro-segmentation. For micro-segmentation the 80/20 rule does no longer apply. It has only to be guaranteed that a switch is able to provide sufficient internal bandwidth.

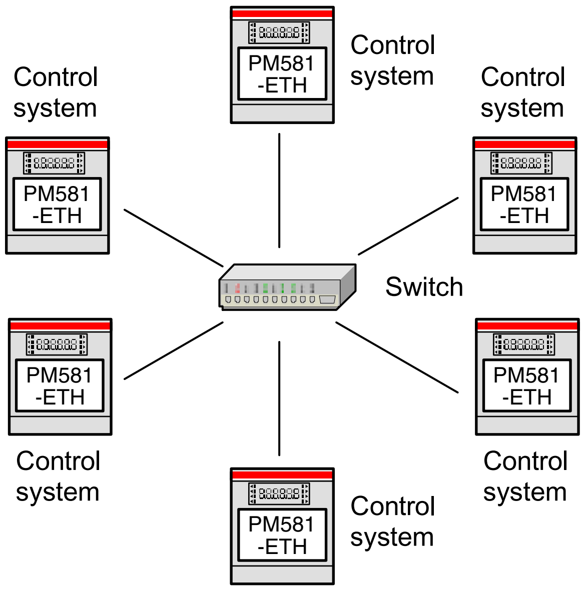


Figure 3-4: Direct connection of all stations to switches

In order to plan a network with optimum performance, we have to think about the question what a network is able to achieve at all. Taking the standards as a basis it can be determined how many data per time can be transmitted via a network theoretically. The smallest Ethernet frame size is 64 bytes long and contains 46 bytes of user data, the maximum frame size is 1518 bytes at 1500 bytes of user data, each plus 64 bits for the preamble and 96 bits for the inter-frame gap. This results in a minimum length of 672 bits (64 x 8 + 64 + 96) and a maximum length of 12304 bits (1518 x 8 + 64 + 96). The transmission of one bit takes 10 ns for fast Ethernet (100 Mbit/s) and 100 ns for Ethernet (10 Mbit/s).

Using these values we can calculate how many data packages of the smallest and the maximum length can be transmitted per second theoretically (see tables). The calculation of the corresponding amount of user data which can be transmitted (without taking into account the additional overheads of the higher protocols) now shows the considerably higher protocol overhead caused by the small data packages.

10 Mbit/s	Length [bits]	Time/bit [ns]	Time/frame [ns]	Frames [ns]	User data/ frame [1/s]	User data [bytes/s]
min. frame	672	100	67 200	14 880	46	684 480
max. frame	12 304	100	1 230 400	813	1 500	1 219 500

Table 3-1: Data rate at 10 Mbit/s

100 Mbit/s	Length [bits]	Time/bit [ns]	Time/frame [ns]	Frames [ns]	User data/ frame [1/s]	User data [bytes/s]
min. frame	672	10	6 720	148 800	46	6 844 800
max. frame	12 304	10	123 400	8 127	1 500	12 195 000

Table 3-2: Data rate at 100 Mbit/s

The corresponding net bandwidth can be calculated from the ratio of the amount of user data per second to the available network bandwidth. The net bandwidth is independent of the transmission rate and calculated in the following table taking a transmission rate of 100 Mbit/s as an example.

100 Mbit/s	**User data [bits/s]**	**Network bandwidth [bits/s]**	**Net bandwidth [%]**
min. frame	54 758 400	100 000 000	54.7
max. frame	97 524 000	100 000 000	97.5

Table 3-3: Net bandwidth at 100 Mbit/s

These calculations point out that the percentage of the network performance is considerably higher for the transmission of larger frames. The efficiency of the data transmission which is independent of the transmission rate is shown in the following table using some selected frame sizes as an example. However, the values given in the table only consider the protocol overhead of the MAC and the network layer. The user data are reduced accordingly by the additional overhead of the corresponding higher layers.

User data [bits]	**Frame size [bits]**	**Overhead [%]**	**Efficiency [%]**
1500	1518	1.2	98.8
982	1000	1.8	98.2
494	512	3.6	96.4
46	64	39.1	60.9

Table 3-4: Efficiency of data transmission

A calculation of the typical transmitted frame sizes may be still possible for small closed networks inside an installation with only automation devices connected. But, for instance, if PCs are additionally connected to the network (even if they are connected only temporarily) the frame sizes can vary considerably. This makes it impossible to perform an exact calculation of the bandwidth or to make a precise statement regarding the performance. However, the following index values could be determined with the help of various studies about network performance.

- For low utilization of 0 to 50 % of the available bandwidth, short response times can be expected. The stations are able to send frames with a typical delay of smaller than 1 ms.
- For medium utilization between 50 and 80 %, the response times can possibly increase to values between 10 and 100 ms.
- For high utilization over 80 %, high response time and wide distribution can be expected. The sending of frames can possibly take up to 10 seconds.

This is why the following principles should be observed when designing an Ethernet network.

- Mixed operation of stations which have to transmit high data volumes and stations which have to operate with short response times (real time) should be avoided. Due to the wide distribution, short response times cannot be guaranteed within such combinations.
- As few as possible stations should be positioned inside of one collision domain. For this purpose, collision domains should be subdivided using switching hubs.

4 Planning examples

4.1 Introduction

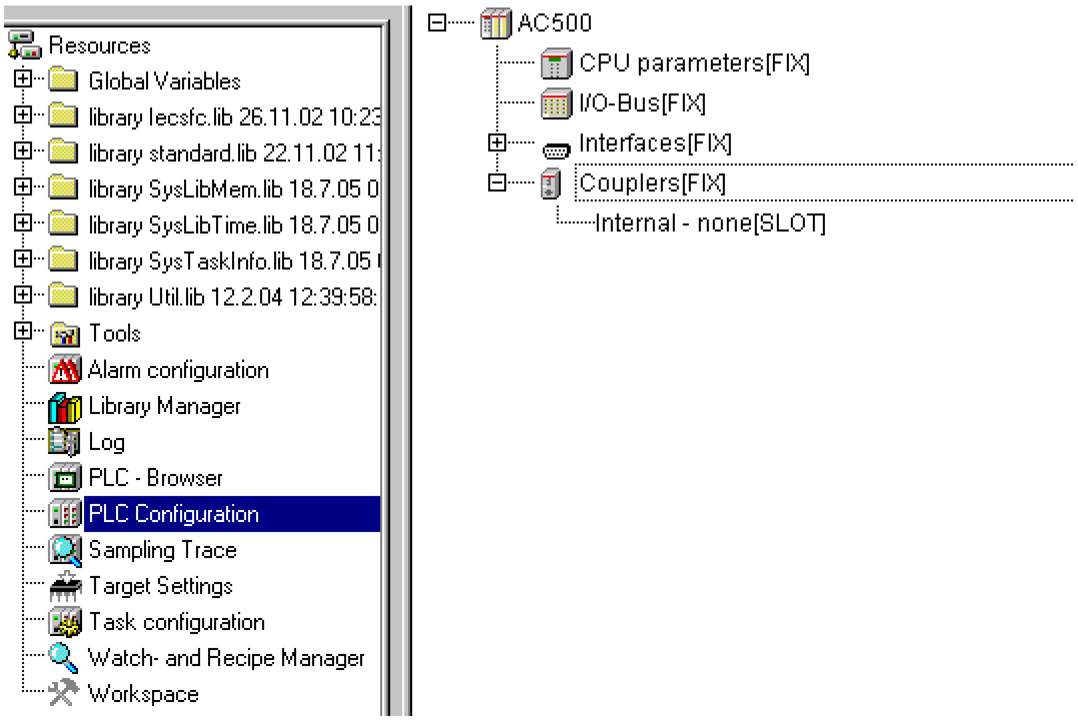
The Ethernet coupler is only completely ready for operation if a valid configuration has been loaded before. Configuring the coupler is always necessary, independent of the intended use of the coupler. Only the parameters to be adjusted depend on the application. Furthermore, some cases require additional implementation within the user program.

In the following sections the general procedure for configuring the Ethernet coupler and the application-specific parameter assignment are explained as well as the possibly required implementation in the user program. In each section regarding the application-specific parameter assignment, one specific application case is described separately. Of course the different functions can also be mixed in any combination.

4.2 Integration of couplers into the controller configuration

First, all couplers to be used in the AC500 have to be specified in the controller configuration (see also System technology of the CPUs / Controller configuration).

To do this, open the "Resources" tab in the left-hand window and then double click on "PLC Configuration".



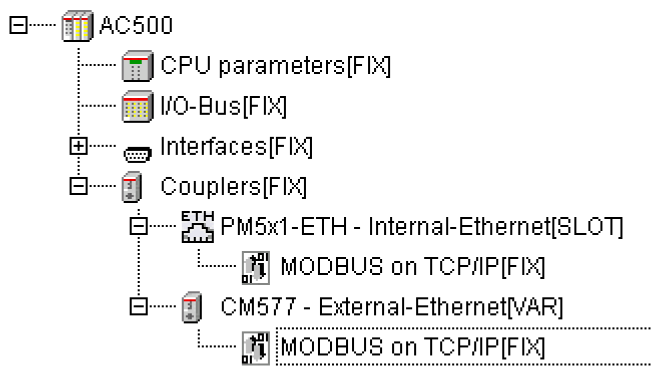
```
Resources
⊞ Global Variables
⊞ library Iecsfc.lib 26.11.02 10:23
⊞ library standard.lib 22.11.02 11:
⊞ library SysLibMem.lib 18.7.05 0
⊞ library SysLibTime.lib 18.7.05 0
⊞ library SysTaskInfo.lib 18.7.05
⊞ library Util.lib 12.2.04 12:39:58:
⊞ Tools
  Alarm configuration
  Library Manager
  Log
  PLC - Browser
  PLC Configuration
  Sampling Trace
  Target Settings
  Task configuration
  Watch- and Recipe Manager
  Workspace
```

```
⊟ AC500
   CPU parameters[FIX]
   I/O-Bus[FIX]
 ⊞ Interfaces[FIX]
 ⊟ Couplers[FIX]
     Internal - none[SLOT]
```

For an internal coupler, select "Internal", press the right mouse button and then select "Replace Element" -> "PM5x1-ETH".

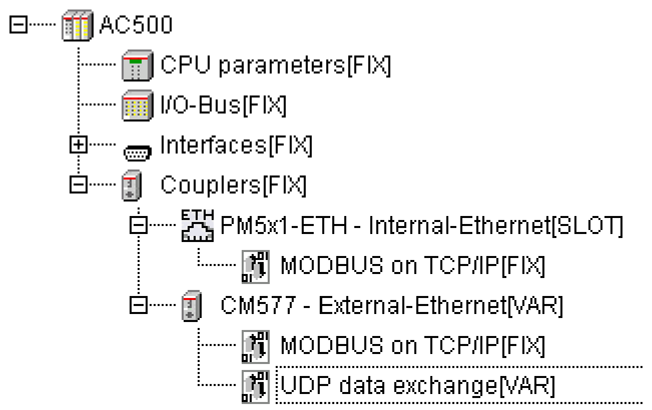
For an external coupler, select "Couplers", press the right mouse button and then select "Append Subelement" -> "CM572".

Please observe that the couplers have to be assigned to the correct slots: The first slot on the left of the CPU is slot 1, the next slot on the left of slot 1 is slot 2, etc. The internal coupler is slot 0.



```
AC500
  CPU parameters[FIX]
  I/O-Bus[FIX]
  Interfaces[FIX]
  Couplers[FIX]
    PM5x1-ETH - Internal-Ethernet[SLOT]
      MODBUS on TCP/IP[FIX]
    CM577 - External-Ethernet[VAR]
      MODBUS on TCP/IP[FIX]
```

The protocol "MODBUS on TCP/IP" is installed together with the couplers by default. If you also want to use UDP, right-click on the corresponding coupler and then select "Append UDP data exchange" from the context menu.



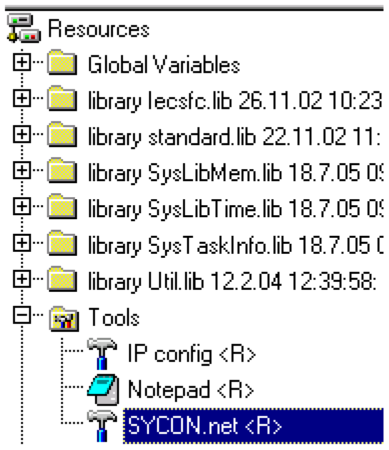
```
AC500
  CPU parameters[FIX]
  I/O-Bus[FIX]
  Interfaces[FIX]
  Couplers[FIX]
    PM5x1-ETH - Internal-Ethernet[SLOT]
      MODBUS on TCP/IP[FIX]
    CM577 - External-Ethernet[VAR]
      MODBUS on TCP/IP[FIX]
      UDP data exchange[VAR]
```

The basic settings of the coupler should not be changed since these settings cover all usual applications.

4.3 General procedure for configuring the coupler

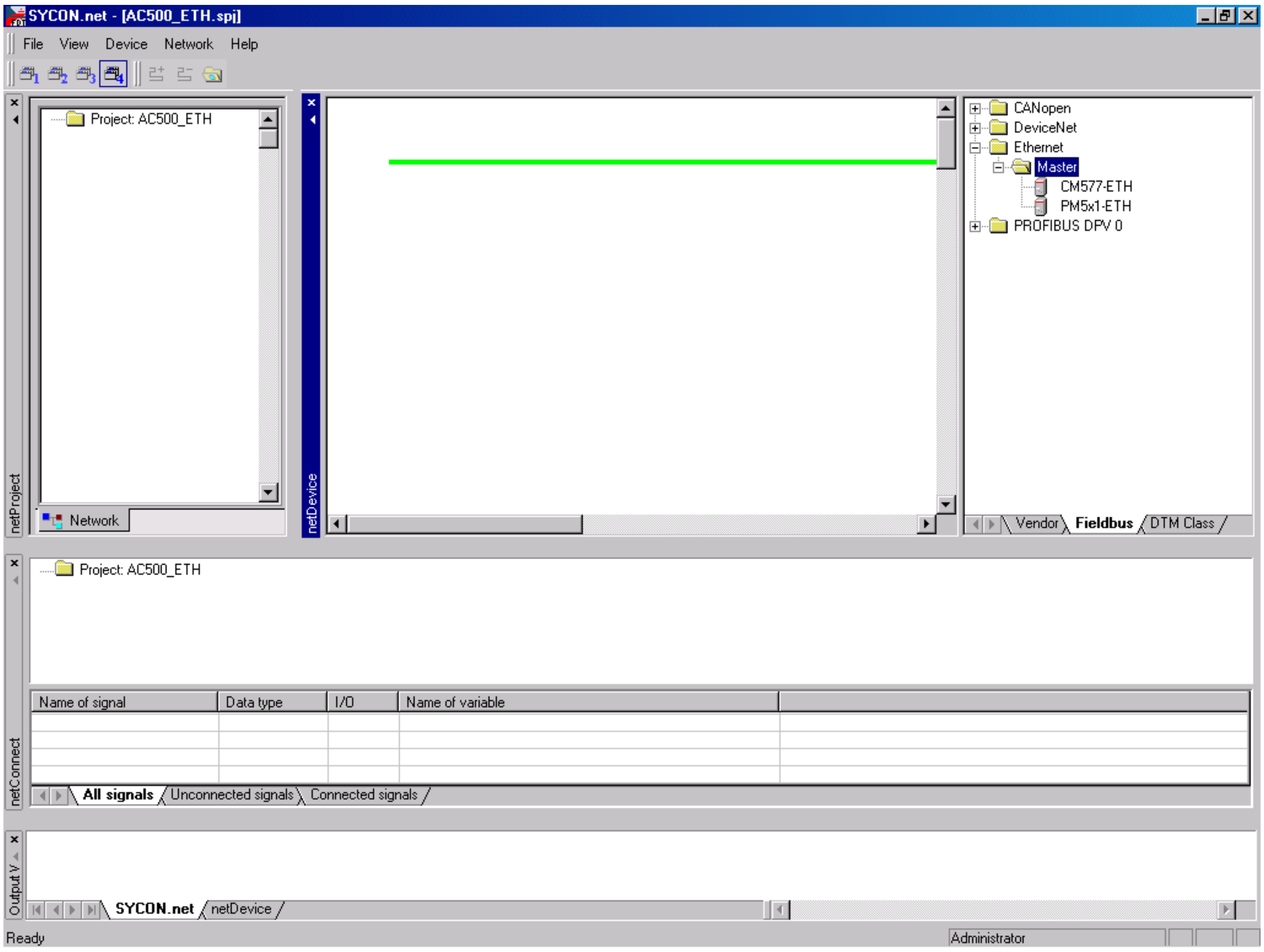
In this section the general procedure for configuring the coupler is described. This procedure has to be performed in any case. All further application-specific parameter assignment is described in the subsequent sections.

To create a configuration file, you first have to start the field bus configuration tool **SYCON.net** from the "Tools" folder of the "Resources" tab in the Control Builder software.



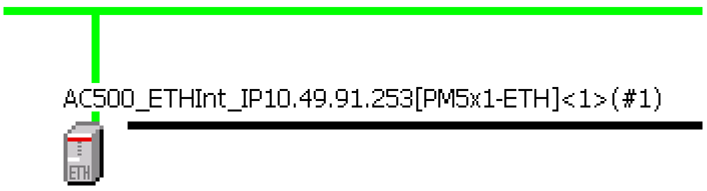
```
Resources
  Global Variables
  library Iecsfc.lib 26.11.02 10:23
  library standard.lib 22.11.02 11:
  library SysLibMem.lib 18.7.05 0
  library SysLibTime.lib 18.7.05 0
  library SysTaskInfo.lib 18.7.05 (
  library Util.lib 12.2.04 12:39:58:
  Tools
    IP config <R>
    Notepad <R>
    SYCON.net <R>
```

The following overview is loaded after the start of the configuration tool SYCON.net.

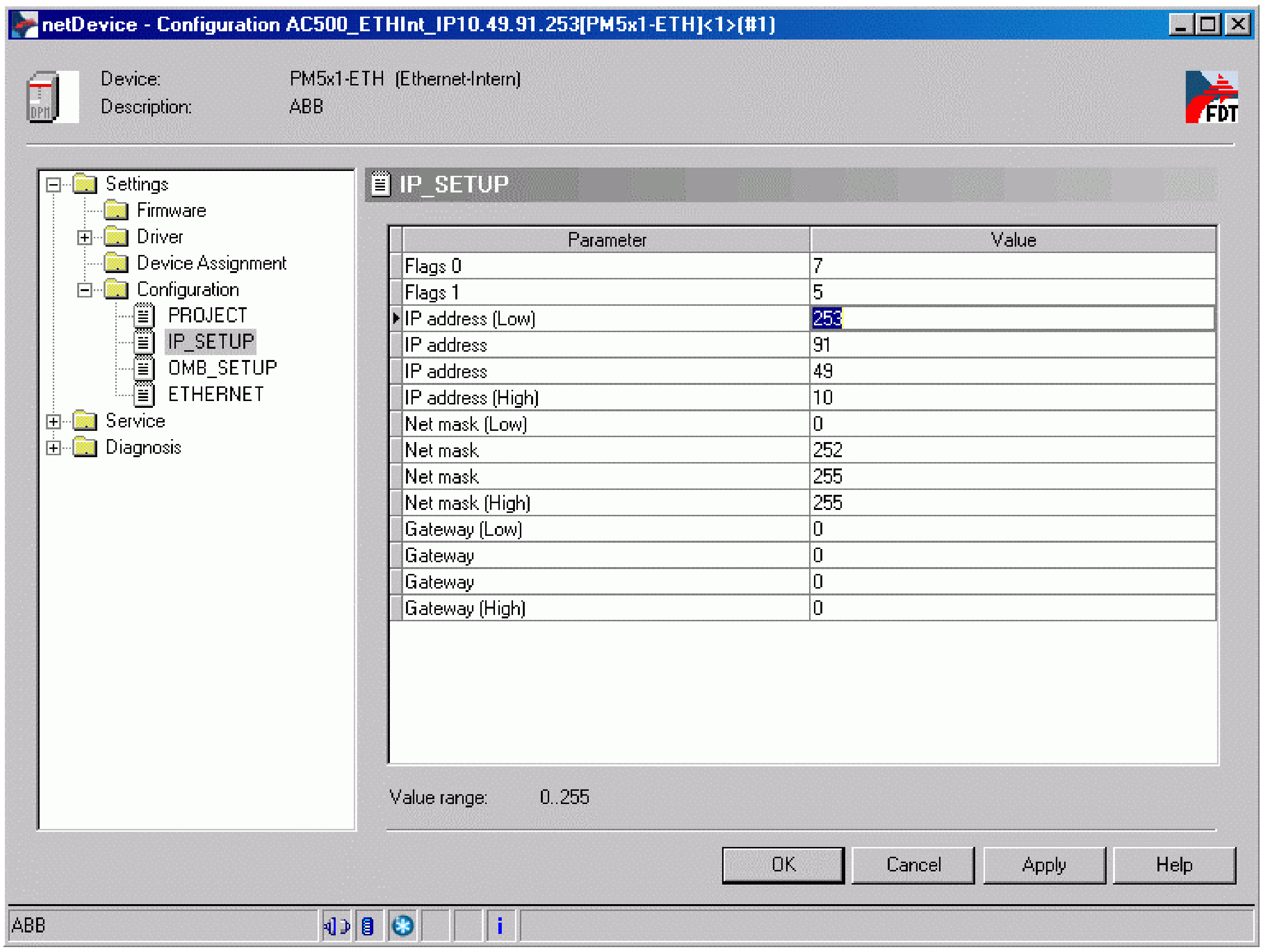


In the right upper window in the register "Ethernet/Master", click on **"PM5x1-ETH"** (internal coupler) or **"CM577-ETH"** (external coupler) and draw it to the green rule into the middle window. With the right position, a **"+"** appears.

Internal coupler



To configure the Ethernet coupler, move cursor on "PM5x1-ETH", then select "right mouse button" and "Configuration".


netDevice - Configuration AC500_ETHInt_IP10.49.91.253[PM5x1-ETH]<1>(#1)

Device: PM5x1-ETH (Ethernet-Intern)
Description: ABB

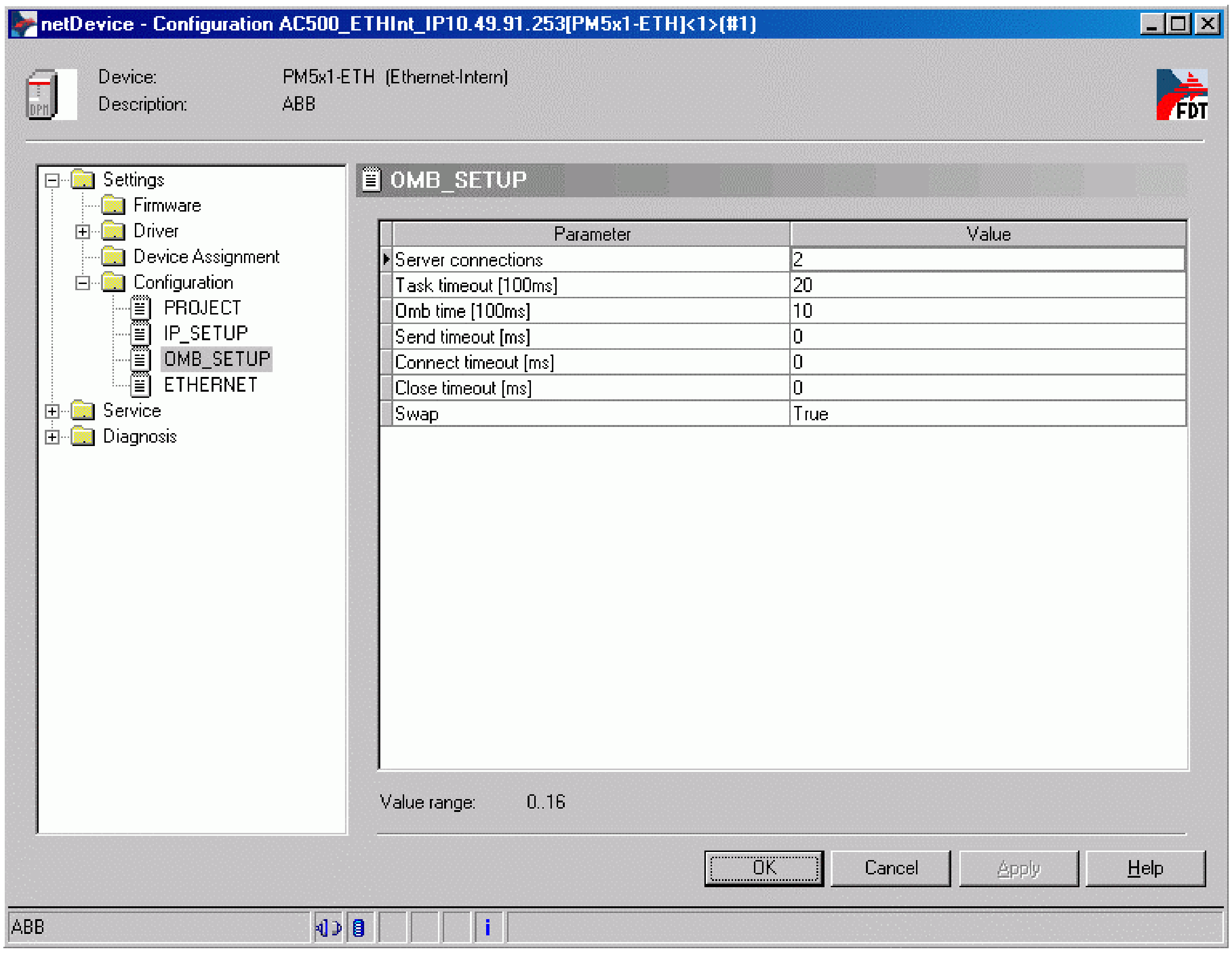
IP_SETUP

Parameter	Value
Flags 0	7
Flags 1	5
IP address (Low)	253
IP address	91
IP address	49
IP address (High)	10
Net mask (Low)	0
Net mask	252
Net mask	255
Net mask (High)	255
Gateway (Low)	0
Gateway	0
Gateway	0
Gateway (High)	0

Value range: 0..255

In the register "Configuration/IP_SETUP" the IP-address, subnet mask and, if necessary, the gateway address is entered (caution: entries begin with the last byte). The values for flag 0 (7 corresponds "to IP ADDRESS available, Netmask available and Gateway available") and flag 1 (5 corresponds "to auto detect and auto negotiate") can be taken over.

If OpenModbus is used, the **"OMB_SETUP"** can be carried out next.



netDevice - Configuration AC500_ETHInt_IP10.49.91.253[PM5x1-ETH]<1>(#1)

Device: PM5x1-ETH (Ethernet-Intern)
Description: ABB

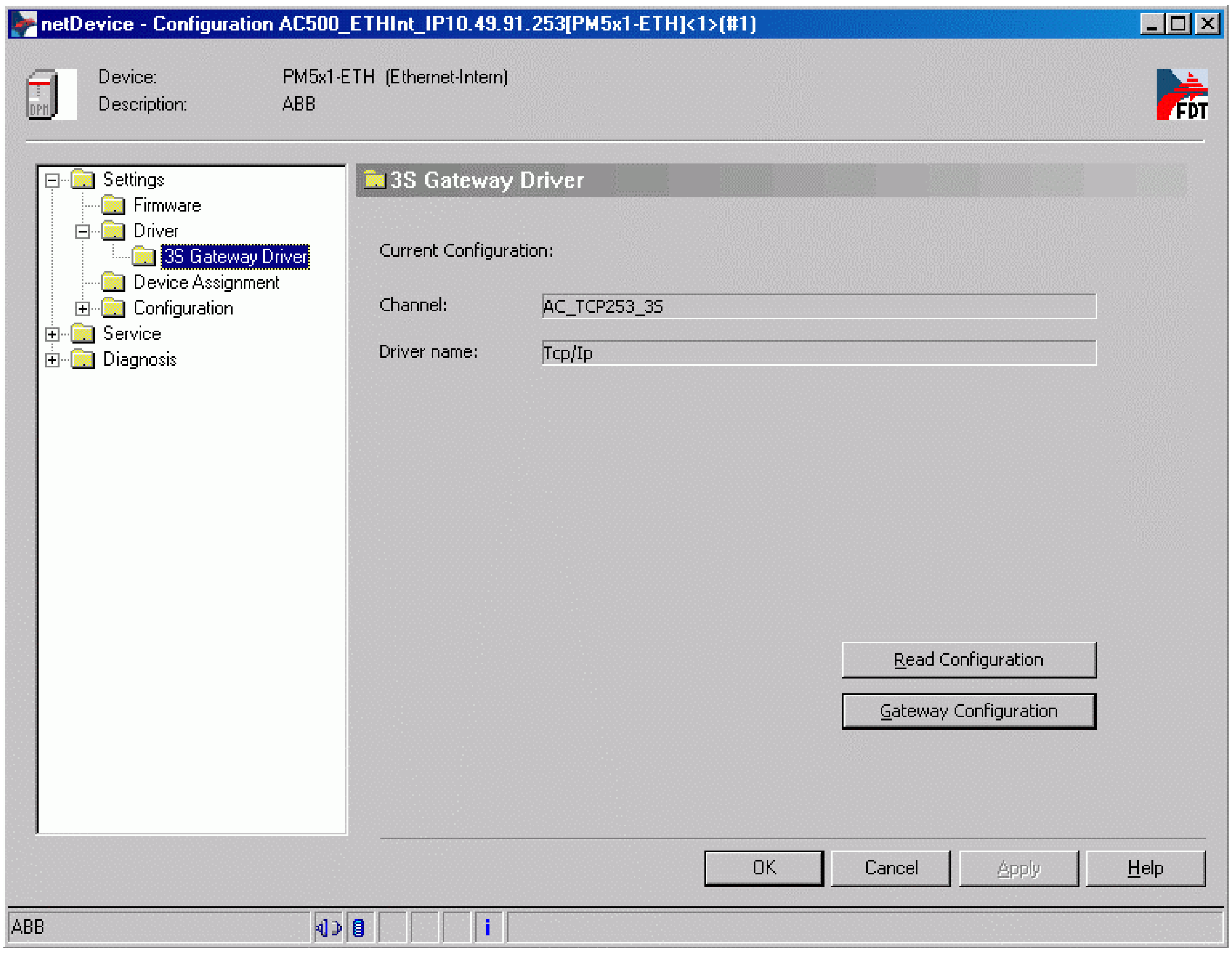
OMB_SETUP

Parameter	Value
Server connections	2
Task timeout [100ms]	20
Omb time [100ms]	10
Send timeout [ms]	0
Connect timeout [ms]	0
Close timeout [ms]	0
Swap	True

Value range: 0..16

In this example, the number of the "server connections" (number of the clients which may access in parallel to a Modbus server (slave)) on 2 and "Swap" is set to TRUE (True corresponds to Motorola byte order).

The configuration can be downloaded now into the Ethernet coupler. To do this, the interface, via which the data has to be loaded into the coupler, must be defined first. The corresponding interface can be configured in the configuration window of the Ethernet coupler in the register "Settings/Driver/3S Gateway Driver".


netDevice - Configuration AC500_ETHInt_IP10.49.91.253[PM5x1-ETH]<1>(#1)

Device: PM5x1-ETH (Ethernet-Intern)
Description: ABB

3S Gateway Driver

Current Configuration:

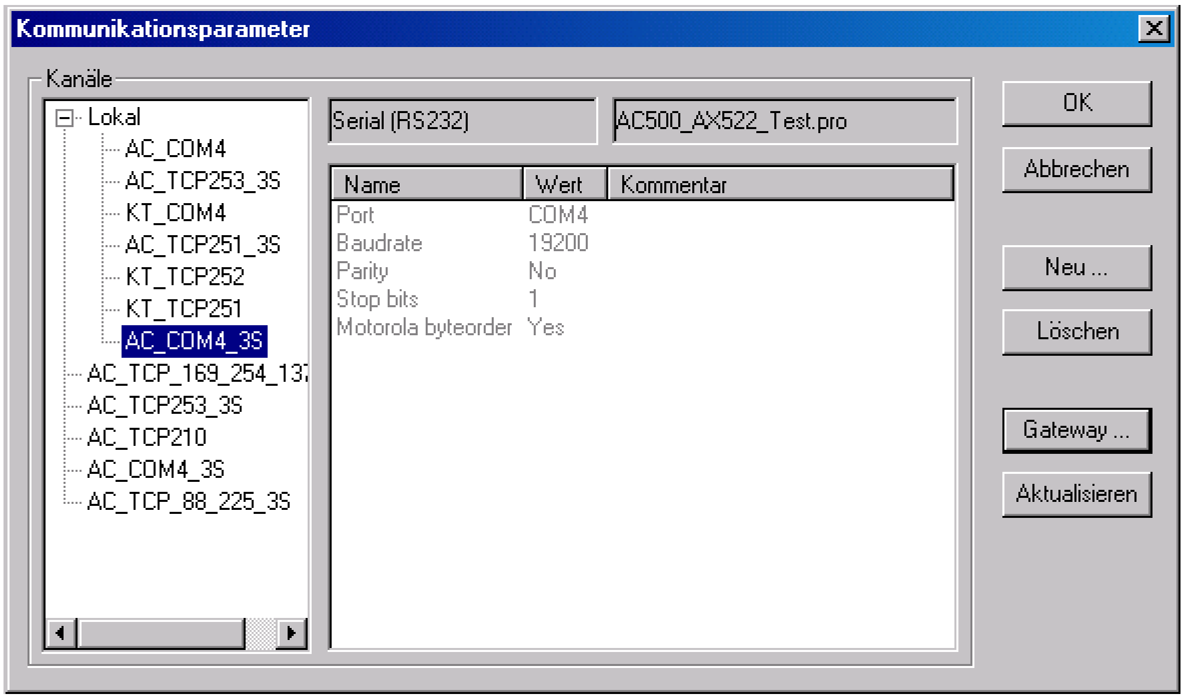
Channel: AC_TCP253_3S

Driver name: Tcp/Ip

In the window on the left, select "Device".

Select the driver "3S Gateway Driver".

Select "Gateway Configuration".

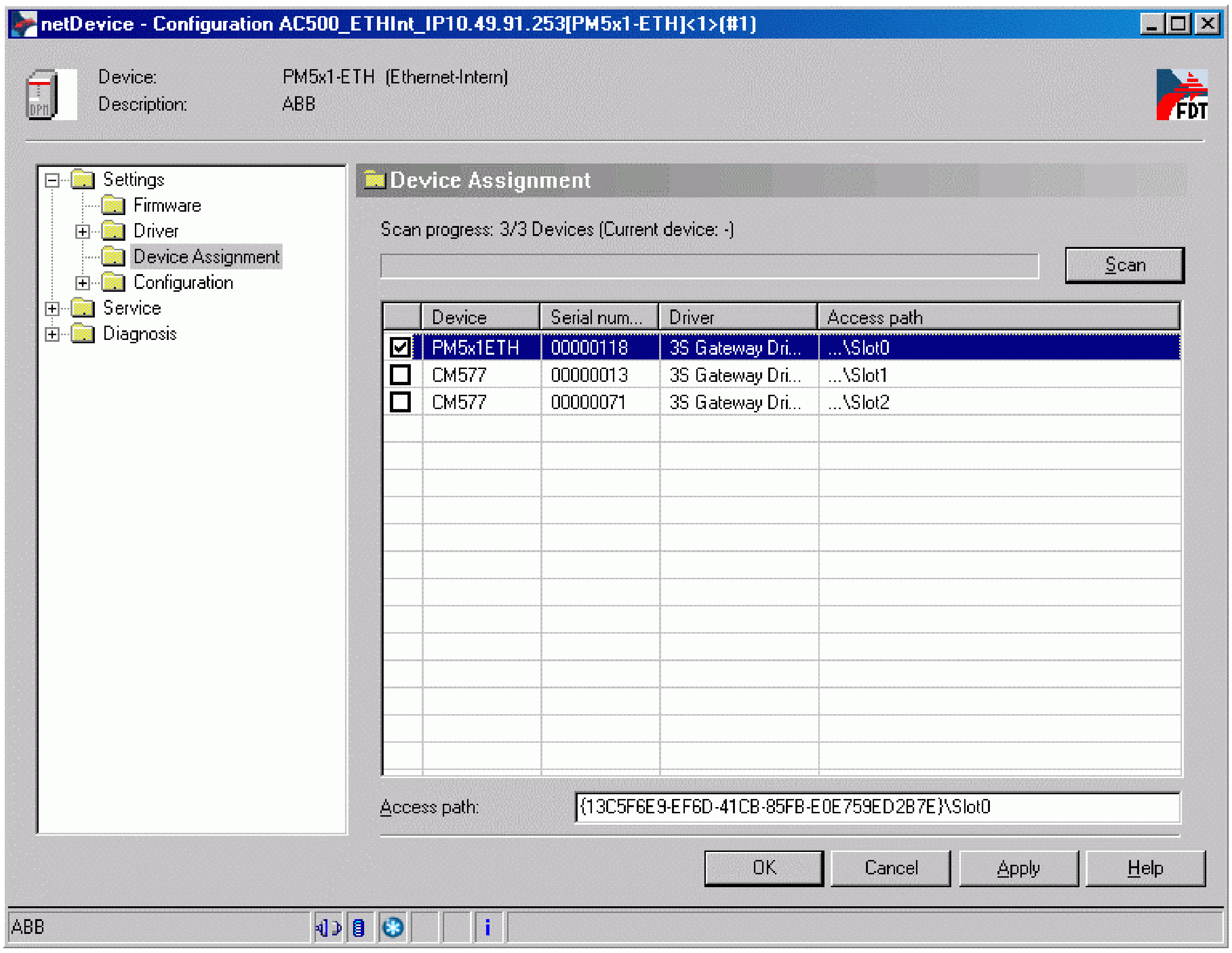

Kommunikationsparameter

Serial (RS232) AC500_AX522_Test.pro

Name	Wert	Kommentar
Port	COM4	
Baudrate	19200	
Parity	No	
Stop bits	1	
Motorola byteorder	Yes	

Select or create new "Gateway", then confirm with "OK".

The configuration tool SYCON.net now searches for Ethernet couplers, which are attached at the given interface. A list of the detected Ethernet couplers can be seen in the configuration window of the Ethernet coupler in the register "Settings/Device Assignment".



Device: PM5x1-ETH (Ethernet-Intern)
Description: ABB

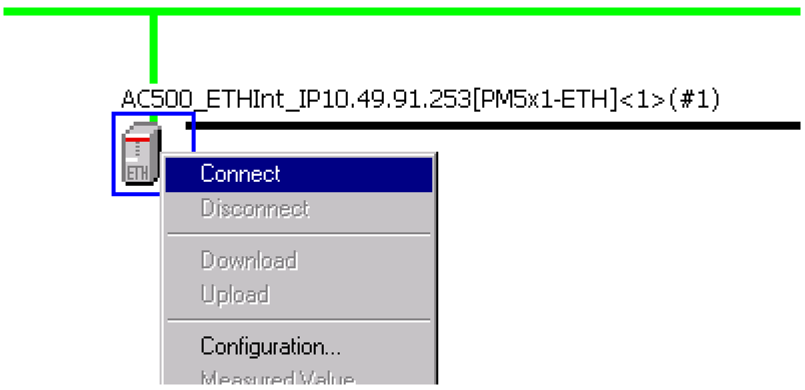
Scan progress: 3/3 Devices (Current device: -)

	Device	Serial num...	Driver	Access path
☑	PM5x1ETH	00000118	3S Gateway Dri...	...\Slot0
☐	CM577	00000013	3S Gateway Dri...	...\Slot1
☐	CM577	00000071	3S Gateway Dri...	...\Slot2

Access path: {13C5F6E9-EF6D-41CB-85FB-E0E759ED2B7E}\Slot0

Select the corresponding coupler and confirm with "OK".

The access to the Ethernet coupler is defined with that. To download the configuration into the coupler, move the cursor in the graphic window on "PM5x1-ETH", then use "right mouse button" and "Connect".

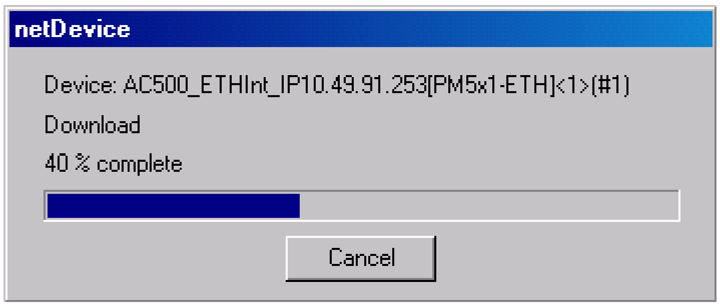


The PM5x1-ETH icon is then highlighted by green background color.

Switch the AC500 to "Stop".

Position the cursor again on the PM5x1-ETH icon in the graphic window, press the right mouse button and select "Download".

Confirm the appearing inquiry dialog with "Yes" to start the download process for the configuration of the internal Ethernet coupler.



Device: AC500_ETHInt_IP10.49.91.253[PM5x1-ETH]<1>(#1)

Download

40 % complete

After a successful download, the coupler is configured.

In the graphic window, position the cursor on the PM5x1-ETH icon, press the right mouse button and select "Disconnect".

The configuration of the internal coupler is now completed.

External coupler

The configuration of the external Ethernet coupler has to be carried out equivalently for the configuration of the internal Ethernet coupler. The distinction between an internal or an external Ethernet coupler is carried out at the beginning of the configuration by choosing the coupler "CM577-RTH".

4.4 Programming access via Ethernet

For programming access via Ethernet the same gateway can be used as for the configuration of the second Ethernet controller. This gateway has to be selected in the CoDeSys software under "Online / Communication Parameters".

4.5 MODBUS on TCP/IP

For the standardized (Open)Modbus protocol two operating modes are distinguished. Controllers with Ethernet coupler can be operated as Modbus on TCP/IP client (master) as well as as server (slave). Simultaneous operation as client and server is also possible. In all operating modes, several Modbus on TCP/IP connections can be provided simultaneously. The maximum possible number of simultaneous connections is only limited by the number of available sockets. Here it has to be observed that each additional communication connection aside from Modbus on TCP/IP also requires one or more sockets.

Operation of the controller as Modbus server only requires a corresponding configuration for the Ethernet coupler to be set up. Operation as Modbus client furthermore requires the implementation of blocks in the user program.

Caution: The user program should not be implemented as a cyclic task (PLC_PRG) if high communication traffic is expected due to the operation as Modbus on TCP/IP client and/or server. A task configuration has to be implemented instead with the cycle time to be adjusted in a way that sufficient communication resources are available in addition to the actual processing time for the program.

4.5.1 Server / slave operation

A typical application for the operation of a controller as (Open)Modbus on TCP/IP server is the linking of an operating terminal via Ethernet. With this application, the operating terminal operates as Modbus client and sends telegrams for reading or writing variables to the controller which executes them accordingly. Since (Open)Modbus on TCP/IP is a standardized protocol, every device supporting this protocol can be connected this way independent of the device type and manufacturer.

For our design example a closed installation-internal network is used. The network consists of four subscribers, two operating terminals and two controllers. The devices are connected to each other via a switch. The operating terminals shall have the IP addresses 10.49.91.251 and 10.49.91.252. The controllers shall have the IP addresses 10.49.91.253 and 10.49.91.254. Both operating terminals shall be able to access both controllers. Since the terminals are not time-synchronized, simultaneous access of both terminals to the same controller must be possible.

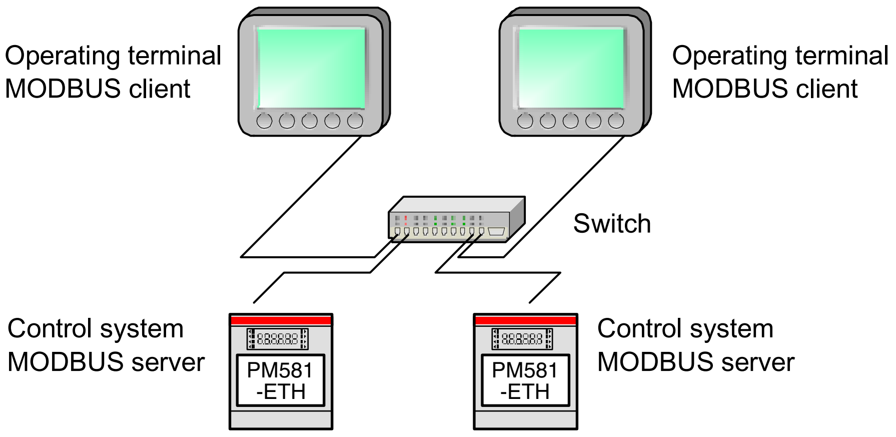


Figure 4-1: Example for OpenModbus on TCP/IP - controller operated as server

Designing the operating terminals as Modbus clients is not subject of this description. For information about this, please contact the corresponding manufacturer. Please refer to the description of the **ETH_MOD_MAST** function block contained in the AC500 documentation for the cross-reference list for client access to the operands of the controller.

⚠ **Caution:** The specified number of parallel server connections results in a permanent reservation of a corresponding number of sockets for these connections. These sockets cannot be used by other protocols. If many clients are used that shall be able to access the controller simultaneously, this can result in a lack of sockets for other protocols. In this case the maximum possible number of simultaneous server connections has to be reduced.

Now, repeat the configuration for the second controller using a different IP address of e.g. 10.49.91.254. All other parameter settings are identical to the settings for the first controller.

Implementation in the user program:

The design process for Modbus server operation is now completed. Since the protocol is automatically processed by the controller, no further blocks in the user program are required. If necessary, the function block **ETH_MOD_INFO** can be inserted into the project for diagnosis. The function block is contained in the library **Ethernet_AC500_V10.lib** (or higher version).

4.5.2 Client / master operation

One possible application for the operation of a controller as (Open)Modbus on TCP/IP client is for example the linking of sensors with Ethernet connection supporting the Modbus protocol. With this application the controller operates as Modbus client and sends telegrams for reading or writing data to the corresponding sensor which in turn generates a corresponding answer. Since (Open)Modbus on TCP/IP is a standardized protocol, every device supporting this protocol can be connected this way independent of the device type and manufacturer.

For our design example a closed installation-internal network is used. The network consists of three subscribers, one controller and two temperature sensors with Ethernet connection. The devices are connected to each other via a switch. The controller shall have the IP address 10.49.91.253. The sensors shall have the IP addresses 10.49.91.251 and 10.49.91.252.

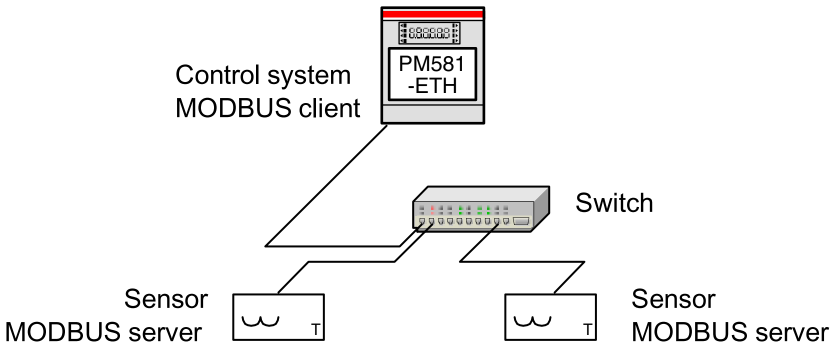


Figure 4-2: Example for OpenModbus on TCP/IP - controller operated as client

Planning the sensors (setting the IP address) as Modbus server is not subject of this description. For information about this, please contact the corresponding manufacturer.

The parameters **Task timeout** and **Omb time** are relevant for controller operation as Modbus client and were already set in the OMB SETUP.

The **Task timeout** parameter determines how long the client shall wait for the server's answer after the transmission of the request to the server before the process is aborted with an error message. For the network considered in our example the default value of 20 x 100 ms can be maintained or even reduced. However, for larger networks with high utilization it can be necessary to increase this value. The valid values for the task timeout parameter range from 1 x 100 ms to 60000 x 100 ms. Here it has to be observed that the controller normally cannot access a server either until it has received its answer or until the timeout has expired. This is why the task timeout value should be dimensioned in a way that no abortion occurs even during temporarily higher utilization. On the other hand, the time reserve should not be too high in order to enable the detection of communication errors as early as possible for a correspondingly high connection performance. For this example a task timeout value of 20 x 100 ms (default) is selected.

The **Omb time** specifies how long the connection shall be maintained after reception of the response telegram. Modbus is based on TCP/IP. One characteristic of this protocol is that a logical communication connection is first established, then the data are exchanged and finally the connection is closed again. The process for establishing and closing the connection takes some time. If data communication between the controller and the servers shall only be performed in longer intervals it makes sense to close the connection immediately after data communication is finished. This avoids unnecessary long blocking of the Modbus access. For fast cyclic data exchange between the client and the server the frequency of connection establishment and closing can be reduced by setting a higher "omb time".

Caution: It has to be observed that the concerning Modbus access (socket) is blocked for other clients as long as the connection is established. If a Modbus server only has a logical access, this results in the fact that no other client can access this server during this time. As long as the connection is established, it also seizes a socket on the controller side. Under certain circumstances this can lead to the fact that not enough sockets can be made available for other protocols during this time.

For this example it is committed that the controller operating as Modbus client shall request the temperature every second from the sensors operating as Modbus servers. Since no other client is accessing the sensors, the connections can be kept open for 1 second. The remain open time is restarted for each connection after each reception of a response from the corresponding server. Therefore, please enter the value 10 x 100 ms here.

The three TCP parameters Send timeout, Connect timeout and Close timeout are related to the TCP protocol used by Modbus. They are considered for client operation as well as for server operation. In the corresponding input fields the respective times have to be specified in milliseconds where 0 is the respective default timeout value. Valid values are 0 to 2.000.000.000.

The **Send timeout** parameter determines how long the controller shall attempt to send a request to a server. Normally the default value 0 which corresponds to 31 seconds can be kept.

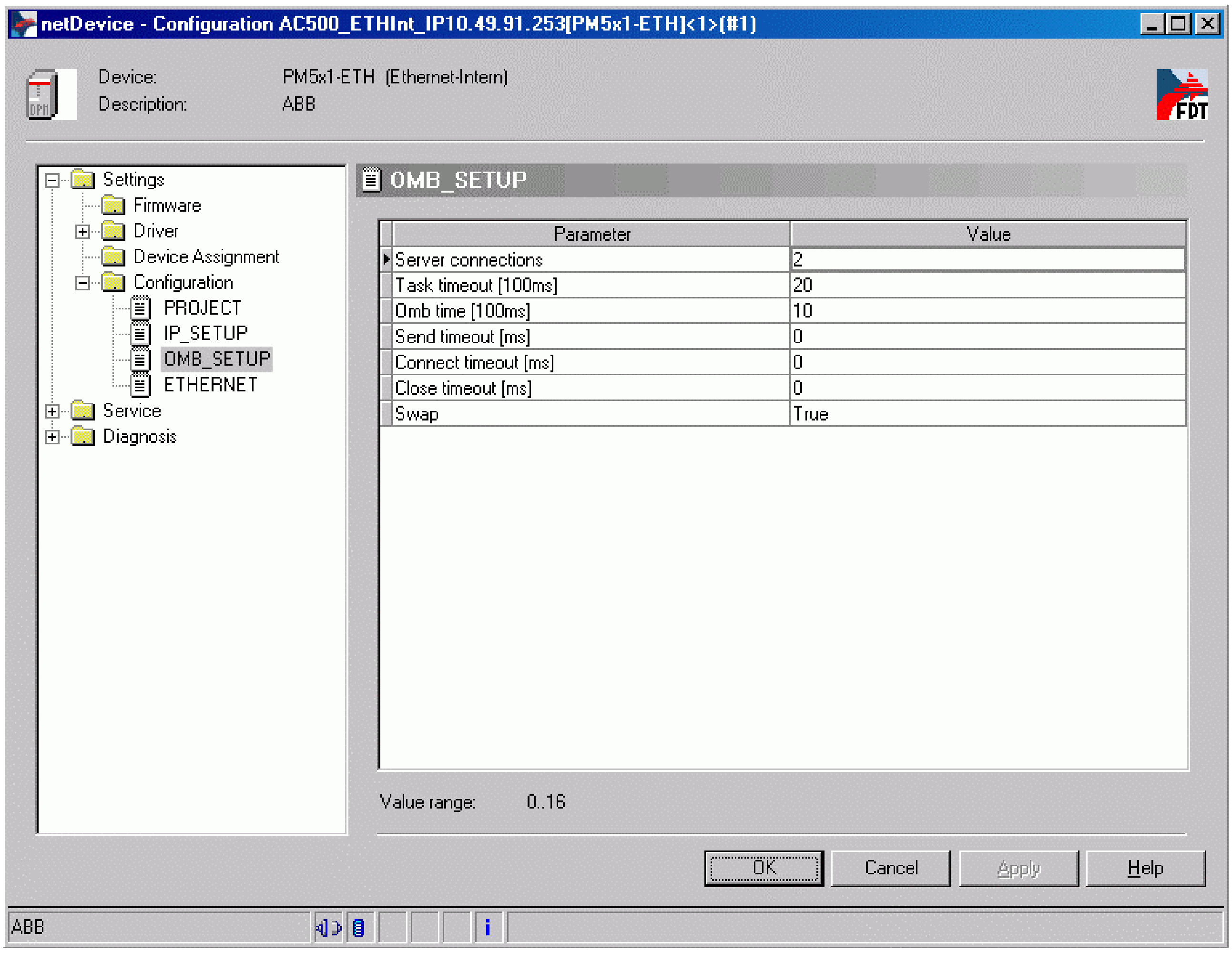
The **Connect timeout** parameter determines how long the controller shall attempt to establish a TCP connection to a server. Normally the default value 0 which corresponds to 31 seconds can be kept.

The **Close timeout** parameter determines how long the controller shall attempt to close a TCP connection to a server. Normally the default value 0 which corresponds to 13 seconds can be kept.

The **Swap** parameter is also relevant for both the client operation and the server operation. With this parameter you can determine whether the two bytes in the words shall be swapped automatically during the transmission of data words. Swapping the word data can be necessary if devices with different processor types are used. If, for example, a server is operating in Motorola format, the word data from the operand memory sent by the controller have to be converted from Intel format into Motorola format (Swap = TRUE) prior to the actual transmission to the server. In the opposite direction the word data of the server are converted into Intel format before they are written to the operand memory of the controller.

Since all subscribers used in this example are processing the data in Motorola format, the Swap parameter has to be set to TRUE.

Thus, the Open Modbus parameter settings are as follows:



Device: PM5x1-ETH (Ethernet-Intern)
Description: ABB

OMB_SETUP

Parameter	Value
Server connections	2
Task timeout [100ms]	20
Omb time [100ms]	10
Send timeout [ms]	0
Connect timeout [ms]	0
Close timeout [ms]	0
Swap	True

Value range: 0..16

The creation of the configuration data is now completed. Continue as described in section "4.3 General procedure for configuring the coupler" and download the configuration data to the Ethernet coupler.

Implementation in the user program:

The coupler is now ready for Modbus client operation. Now the blocks for starting the requests to the servers have to be implemented into the user program.

For this purpose, start the CoDeSys programming software and create a new project or open a corresponding existing project. At first integrate the used Ethernet coupler in the PLC configuration. Insert an instance of the function block **ETH_MOD_MAST** in your project now.

Assign the slot number (module number) of the used Ethernet coupler to the block input **Slot**. If the internal Ethernet coupler is used, this corresponds to module number 0. When using external couplers, these couplers are numbered consecutively from right to left. The first external coupler has the module number 1.

The IP address of the first sensor is 10.49.91.251. This IP address has to be applied to input **IP_ADR** of the MODMAST block. Each byte in IP_ADR represents one octet of the address. Thus, for our example the value 16#0A315BFB (hexadecimal) or 171006971 (decimal) has to be assigned to IP_ADR.

Since the temperatures shall be read from the sensors in the form of a word, the value 3 has to be assigned to **FCT** and 1 has to be specified for **NB**. At input **ADDR**, the register address in the server has to be specified from which the word should be read. Starting from the assumption that the current temperature is stored in the sensor under register address 0, you have to enter the value 0 here.

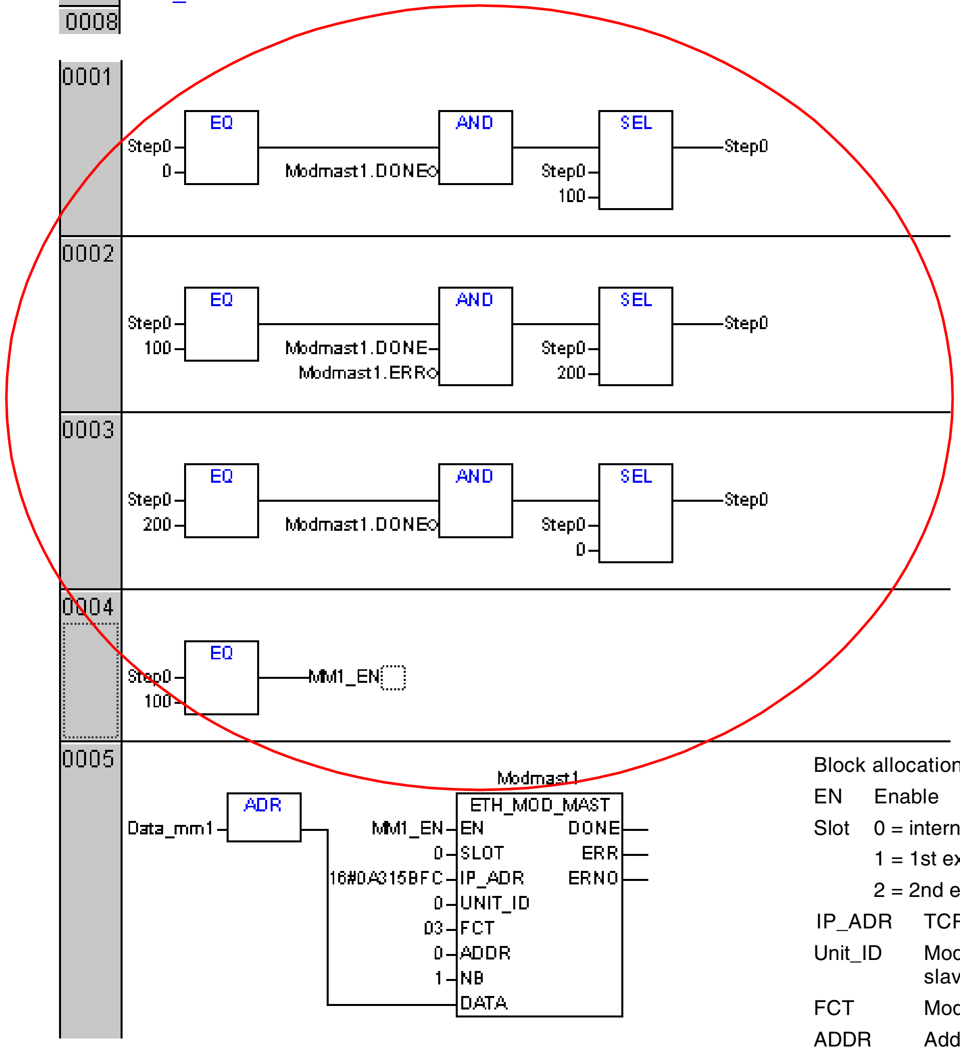
Furthermore, the data read from the first sensor shall be stored in the variable **Data_MM1** in the operand memory of the controller. Thus, block input **DATA** has to be wired with this operand via the ADR block.

The Modbus request to a server is started with a FALSE &ndash;> TRUE edge at input **EN**. The release of input **EN** is controlled by a step chain depending on the outputs **DONE** and **ERR** (available for exactly one PLC cycle after the task execution).

Represented in FBD, this results in the following program part for the first server.

```
PROGRAM MM1
VAR
    Modmast1: ETH_MOD_MAST;
    Step0: INT;
    MM1_EN: BOOL;
    Data_mm1: INT;
END_VAR

```

Declaration of variables



Step chain to control the ETH_MOD_MAST function block.

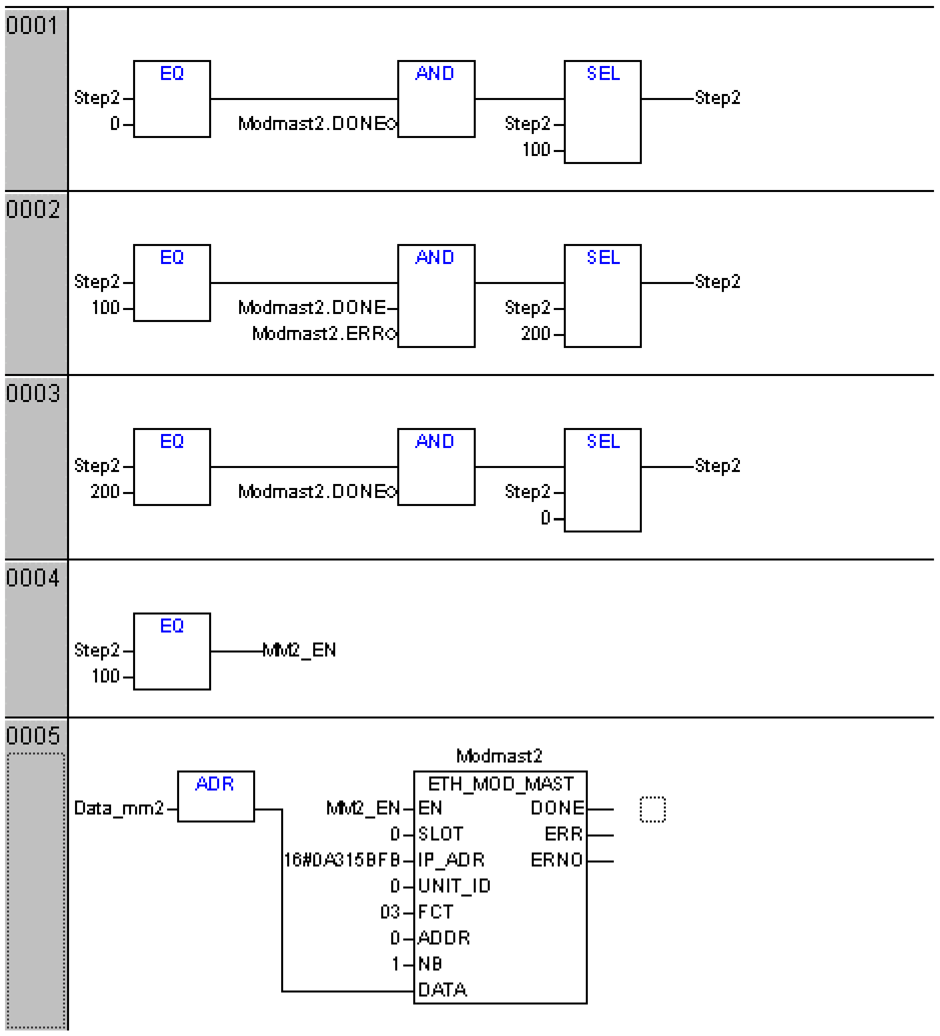
The enable of the next task is delayed as long as the old task has not been completed without errors (the DONE output is TRUE for one cycle, the ERR output must be FALSE during this period of time).

Block allocation

EN	Enable	
Slot	0 = internal	
	1 = 1st external slot	
	2 = 2nd external slot	
IP_ADR		TCP/IP address of the receiver
Unit_ID		Modbus slave address of a serial slave (if a gateway is used)
FCT		Modbus Function Code
ADDR		Address in the memory of the receiver
NB		Number of bytes
DATA		Location of data in the master

Now insert a corresponding block for the second server. The assignments for the block inputs are almost identical. Only the IP address (16#0A315BFC), the operand used to store the read values (Data_MM2) and the step chain are different.

```
PROGRAM MM2
VAR
    Modmast2: ETH_MOD_MAST;
    Step2: INT;
    MM2_EN: BOOL;
    Data_mm2: INT;
END_VAR
```



If required, the program additionally has to be expanded by an evaluation of possible errors. If an error occurs during the processing of a Modbus request, this is indicated at output ERR of the corresponding ETH_MODMAST block. The general Modbus processing status can be displayed and evaluated using the ETH_MODSTAT block.

Download the project to the controller. After this, implementation of the Modbus client is completed.

4.6 Fast data communication via UDP/IP

4.6.1 Example configuration for data communication via UDP/IP

Fast data communication via UDP/IP is a proprietary (manufacturer-specific) protocol. This protocol is only supported by AC31 series 90 or AC500 controllers. It serves for the transmission of any data between the controllers and can be used in parallel to all other protocols. This protocol is based on the standardized protocols UDP and IP. The actual protocol is lying above UDP/IP.

Protocol handling corresponds to a large extent to the ARCNET processing. Aside from the corresponding configuration of the Ethernet coupler, the implementation of the following bocks is required for fast data communication via UDP/IP:

- ETH_UDP_REC
- ETH_UDP_SEND
- ETH_UDP_STO

Caution: The user program should not be implemented as a cyclic task (PLC_PRG) if high communication traffic caused by the protocol for fast data communication via UDP/IP is expected. A task configuration has to be implemented instead with the cycle time to be adjusted in a way that sufficient communication resources are available in addition to the actual processing time for the program.

For our design example a closed installation-internal network is used. The network consists of two AC500 controllers. The devices are connected to each other via a switch. The controllers shall have the IP addresses 10.49.91.253 and 10.49.91.254. Each controller shall send 2 bytes of data to the other controller. The uniform and constant user data length has been chosen only to simplify the example. If necessary, it is also possible to use different data lengths for each controller as well as variable data lengths for each transmission.

The individual transmissions shall be controlled in different ways for each controller.

Controller 1 (IP address 10.49.91.253)
transmits the data to the other controller periodically and receives data from the other controller.

Controller 2 (IP address 10.49.91.254)
transmits the data to the other controller periodically and receives data from the other controller.

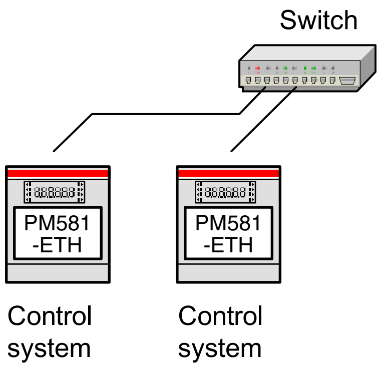


Figure 4-3: Example configuration for data communication via UDP/IP

4.6.2 Configuring the Ethernet couplers for data communication via UDP/IP

The configuration of the couplers has to be performed according to section 4.3.

4.6.3 Implementation in the user program

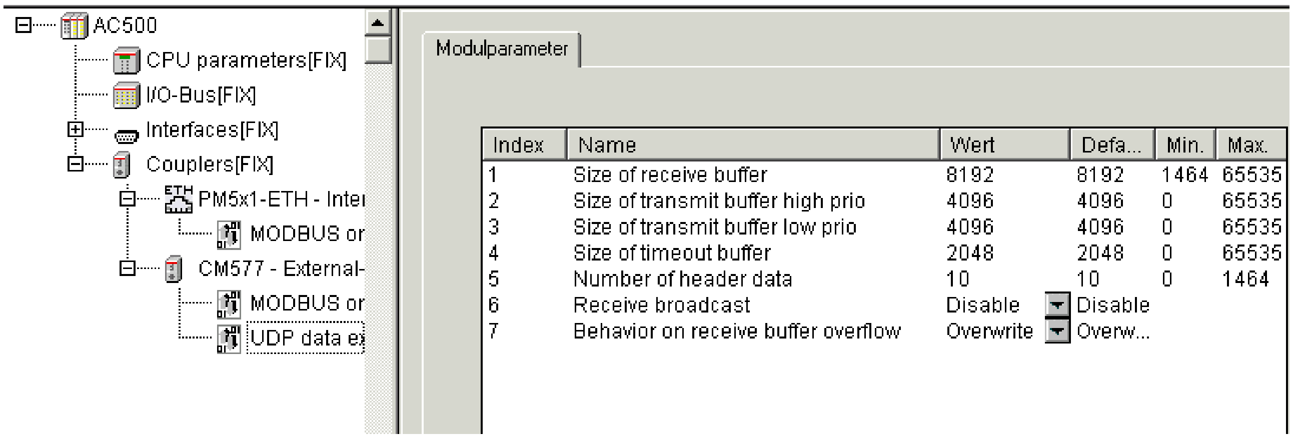
Configuring the couplers for fast data communication via UDP/IP is completed. Now the blocks for processing the protocol have to be implemented into the different user programs.

For this purpose, start the CoDeSys programming software and create a new project or open a corresponding existing project. At first integrate the used Ethernet coupler in the PLC configuration.

Protocol processing initialization:

The procedure for initializing the protocol processing is identical for all controllers used in this example.

The initialization of the protocol processing and the required resources is done in the controller configuration.



Index	Name	Wert	Defa...	Min.	Max.
1	Size of receive buffer	8192	8192	1464	65535
2	Size of transmit buffer high prio	4096	4096	0	65535
3	Size of transmit buffer low prio	4096	4096	0	65535
4	Size of timeout buffer	2048	2048	0	65535
5	Number of header data	10	10	0	1464
6	Receive broadcast	Disable	Disable		
7	Behavior on receive buffer overflow	Overwrite	Overw...		

The desired receive buffer length in bytes is specified using the parameter "Size of receive buffer". All received telegrams are first stored in this buffer and can then be read by means of the ETH_UDP_REC block. Aside from the actual user data of a telegram the sender's IP address and the telegram length are stored. The value 8192 bytes should be sufficient for most applications.

The parameters **"Size of transmit buffer high prio"** and **"Size of transmit buffer low prio"** are used to specify the transmit buffer sizes for telegrams with high or low priority in bytes. UDP telegrams can be transmitted either with normal or high priority. For each priority a separate transmit buffer is available. If only one transmit priority is used in the program, the corresponding transmit buffer has to be configured with a sufficient size.

If mixed priorities are used, the telegrams with high priority are transmitted first. Telegrams with normal priority (if available) are not transmitted until the transmit buffer for telegrams with high priority does no longer contain any telegrams. If another message with high priority is written into the transmit buffer before the transmission of all telegrams with normal priority is finished, the normal priority transmission is interrupted for the transmission of the high priority message. After this, the transmission of telegrams with normal priority is continued.

The parameter **"Size of timeout buffer"** is used to specify the length of the timeout buffer. This buffer is used to automatically store information about telegrams which could not be transmitted to the recipient due to an error. Errors can be caused e.g. by an interrupted Ethernet line, an incorrectly specified IP address for the recipient which does not exist in the network or a recipient controller which is in STOP state.

The parameter **"Receive broadcast"** is used to specify whether received broadcast telegrams shall be stored in the receive buffer (Enable) or dismissed (Disable). In our example no broadcasts are used. The parameter is therefore set to 'Disable'.

The parameter **"Behavior on receive buffer overflow"** is used to specify what should happen if further telegrams are received after the receive buffer is full. If the parameter is set to 'Overwrite', the oldest telegrams stored in the buffer are replaced by the received new telegrams. If it is set to 'Reject', the data stored in the buffer are kept and the subsequently received telegrams are dismissed instead until sufficient space is available again in the buffer.

Note: In order to avoid receive buffer overflow e.g. due to higher data traffic than expected, enlarge the receive buffer or read the buffer more frequently via the ETH_UDP_REC block.

Transmission of data:

The transmit process or the process of storing a telegram to be transmitted in the corresponding buffer is initiated by a rising edge applied at input **EN** of the **ETH_UDP_SEND** block. The block confirms this process with a rising edge at output **DONE**. If an error occurred during this process (e.g. receive buffer full), this is indicated at output **ERR**. A telegram to be transmitted was only stored successfully in the corresponding transmit buffer if DONE = TRUE and ERR = 0. The variable to be used to store a transmit telegram has to be applied to block input DATA via the ADR operator.

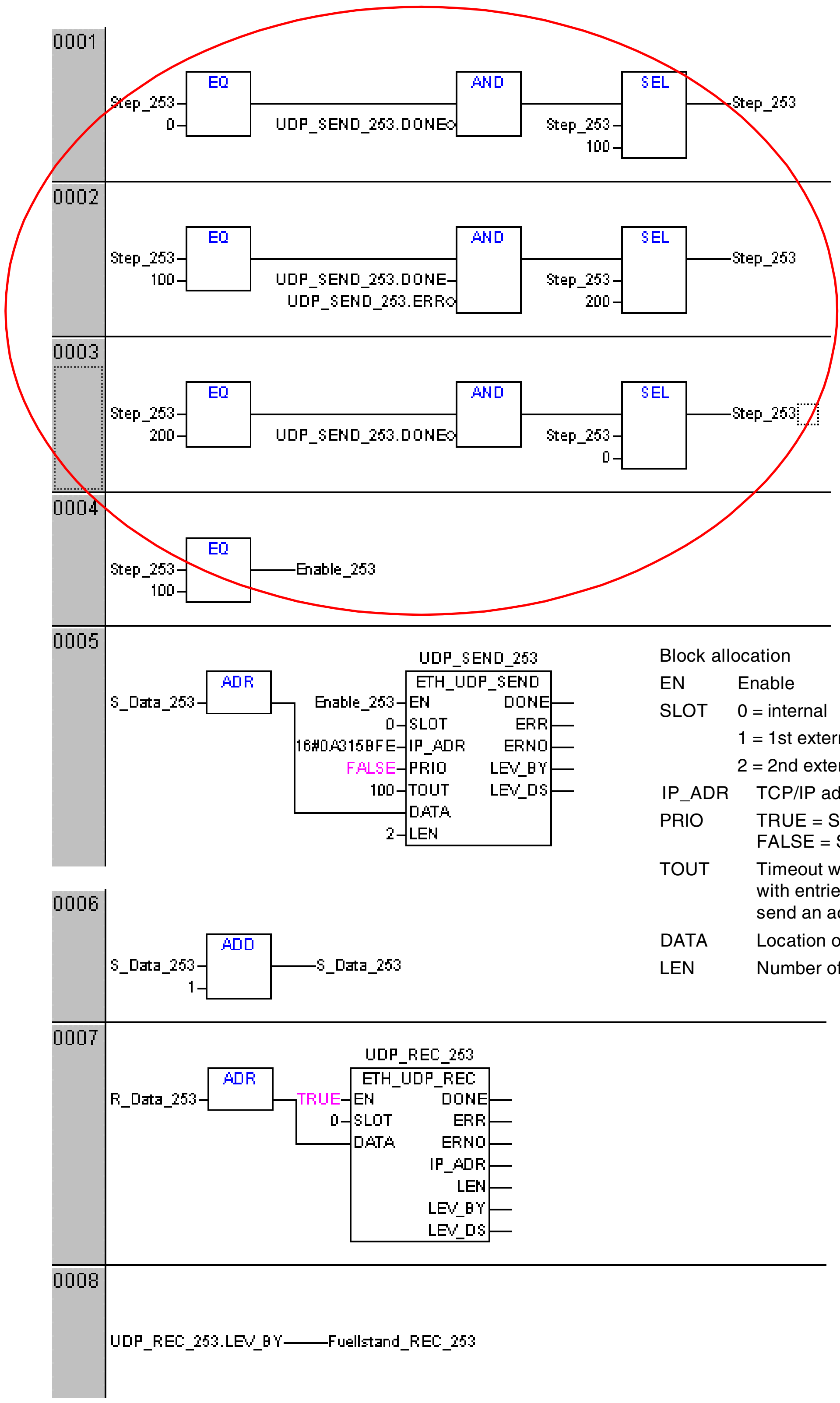
Reception of data:

Reading telegrams from the receive buffer is done using the **ETH_UDP_REC** block. If necessary, it is also possible to use several instances of the ETH_UDP_REC block within one program to enable faster reading of the telegrams. The block is processed if input **EN** = TRUE. The corresponding coupler has to be specified at input **SLOT** (0 = int. coupler, 1 = 1st ext. coupler slot, etc.). The variable to be used to store a receive telegram has to be applied to block input **DATA** via the ADR operator.

The following program can be used for our example (data exchange between two controllers):

```
PROGRAM UDP_253
VAR
    Step_253: INT;
    UDP_SEND_253: ETH_UDP_SEND;
    Enable_253: BOOL;
    S_Data_253: INT := 1;
    UDP_REC_253: ETH_UDP_REC;
    R_Data_253: INT;
    Fuellstand_REC_253: WORD;
END_VAR
```

Declaration of variables



Step chain to control the ETH_UDP_SEND function block.

The enable of the next task is delayed as long as the old task has not been completed without errors (the DONE output is TRUE for one cycle, the ERR output must be FALSE during this period of time).

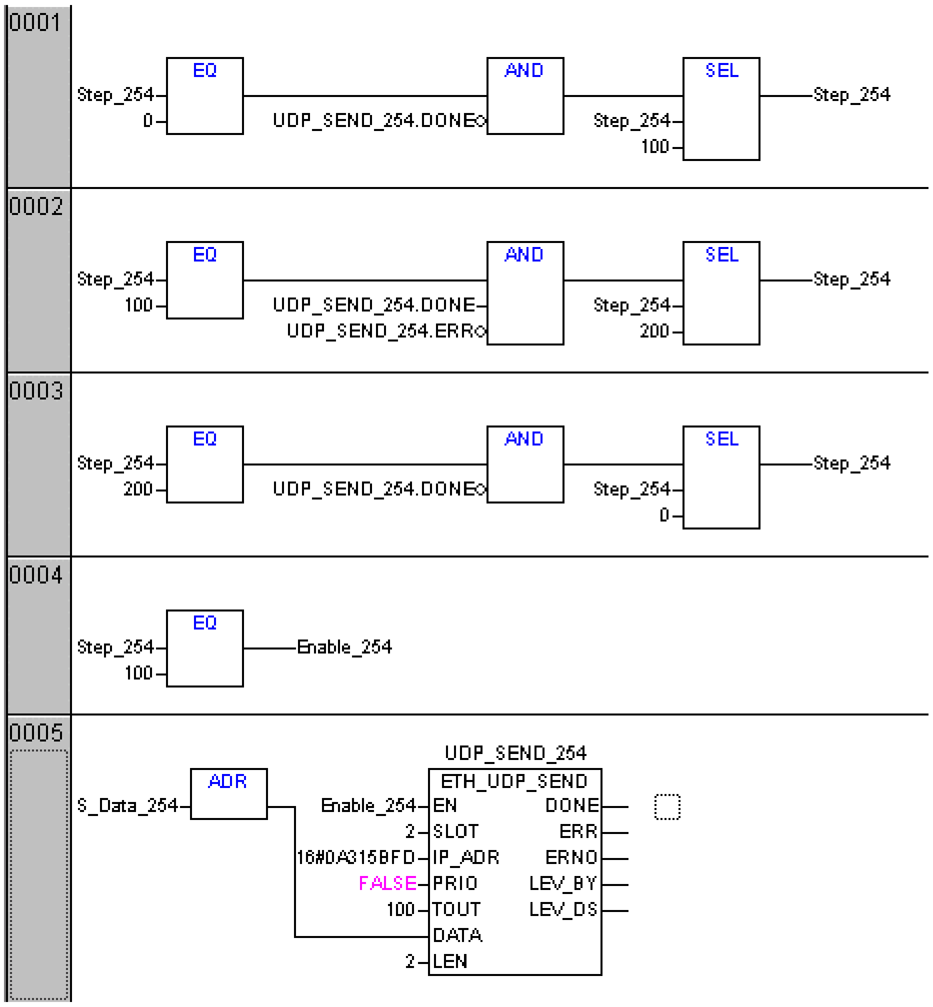
Block allocation

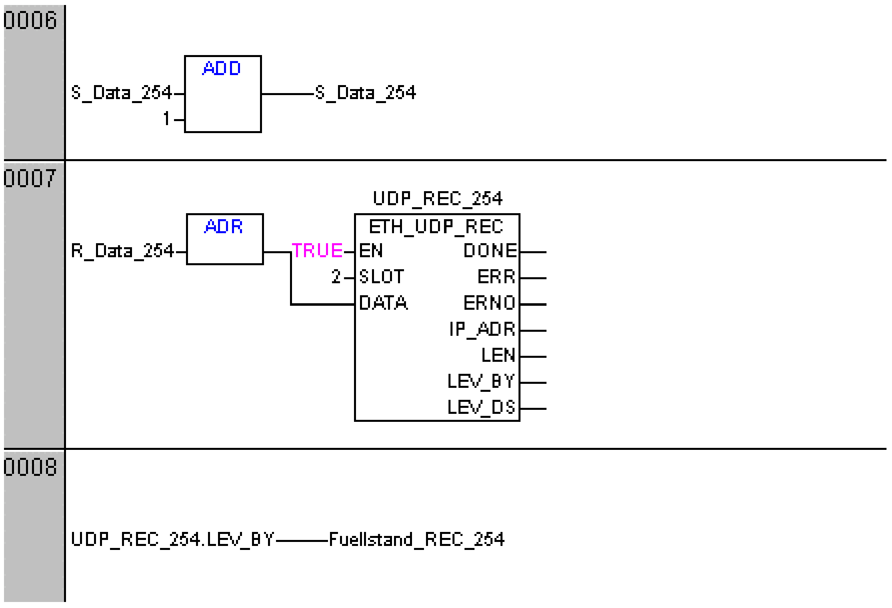
EN	Enable
SLOT	0 = internal
	1 = 1st external slot
	2 = 2nd external slot
IP_ADR	TCP/IP address of the receiver
PRIO	TRUE = Sending with high priority
	FALSE = Sending with low priority
TOUT	Timeout waiting time of the sender, with entries >0, the receiver must send an acknowledgement
DATA	Location of data to be sent
LEN	Number of bytes (length)

The values TRUE at output **DONE** and 0 at output **ERR** indicate that a telegram has been successfully read from the buffer during the present cycle and copied to the area specified at input DATA. The user data length of the telegram is output at **LEN**. The block always reads the next receive telegram stored in the buffer without taking into account the IP address of its sender. The sender's IP address of the respective telegram is output at **IP_ADR**.

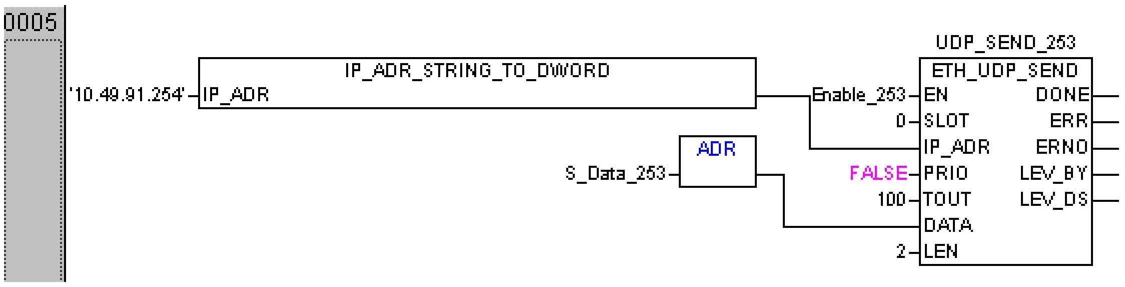
The program for the second controller is almost identical.

```
PROGRAM UDP_254
VAR
    Step_254: INT;
    UDP_SEND_254: ETH_UDP_SEND;
    Enable_254: BOOL;
    S_Data_254: INT := 1;
    UDP_REC_254: ETH_UDP_REC;
    R_Data_254: INT;
    Fuellstand_REC_254: INT;
END_VAR
```





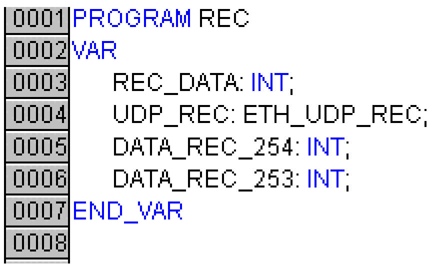
Instead of specifying the TCP/IP address as a hex value at block ETH_UDP_SEND, the function "IP_ADR_STRING_TO_DWORD" contained in library "ETHERNET_AC500_V10.lib" (folder "IP conversions") can also be used. In this case, the address is specified as string.



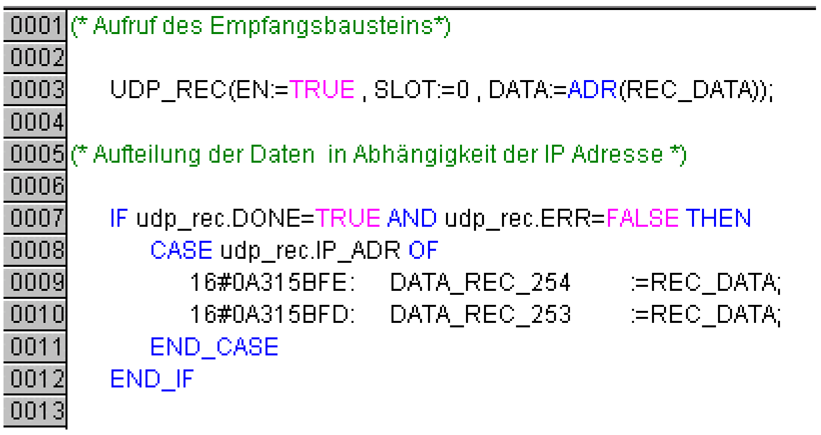
For output IP_ADR of block "ETH_UDP_REC", an inverse function is available in order to obtain the TCP/IP address as string (IP_ADR_DWORD_TO_STRING).

If telegrams are received from different controllers and if the data received from the respective controller shall not be overwritten by the data received from another controller, it is furthermore required to copy the data to a specific area for the sender's IP address.

The implementation of this distribution is easier to realize using an ST program.



```
PROGRAM REC
VAR
    REC_DATA: INT;
    UDP_REC: ETH_UDP_REC;
    DATA_REC_254: INT;
    DATA_REC_253: INT;
END_VAR
```



```
(* Aufruf des Empfangsbausteins*)

    UDP_REC(EN:=TRUE , SLOT:=0 , DATA:=ADR(REC_DATA));

(* Aufteilung der Daten  in Abhängigkeit der IP Adresse *)

    IF udp_rec.DONE=TRUE AND udp_rec.ERR=FALSE THEN
        CASE udp_rec.IP_ADR OF
            16#0A315BFE:    DATA_REC_254        :=REC_DATA;
            16#0A315BFD:    DATA_REC_253        :=REC_DATA;
        END_CASE
    END_IF

```

Here, the data are copied to different areas according to the TCP/IP address using the "CASE" command.

16#0A31B5FE   10.49.91.254

16#0A31B5FD   10.49.91.253

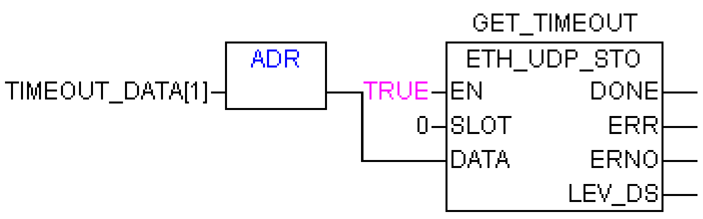
In case of higher amounts of data, complete arrays can be created instead of an integer.

This example can be easily expanded for a higher number of stations.

If a telegram stored in the transmit buffer could not be transmitted to the target controller for any reason or if the target controller did not acknowledge the reception of the telegram within the specified send timeout of 100 ms, the information about this telegram specified during the initialization are stored in the timeout buffer. In case of a buffer overflow, the respective oldest entry is overwritten and the number of entries still stays at the maximum value.

If the individual controllers are started one after the other, it can happen that the value of LEVLTOS first increases in the controllers already started since the controllers, which are not yet in RUN state, are not yet able to process the protocol. Then, after all controllers are in RUN state and if the Ethernet network is connected correctly and not overloaded, the LEVLTOS values should no longer change in any controller.

If necessary, the information about the undeliverable transmit telegrams stored in the timeout buffer can be read via the **ETH_UDP_STO** block and evaluated. The figure below shows the corresponding block call for all controllers.



If output DONE is TRUE and ERR is 0, a data record has been read from the buffer and stored in TIMEOUT_DATA. If necessary, it is now possible to further evaluate the data record. Furthermore, the number of entries in this buffer is decremented if no other timeout occurred in the meantime. According to the initialization of the protocol processing, TIMEOUT_DATA[1] to [4] contain the IP address of the target controller and TIMEOUT_DATA[5] contains the first byte of the original user data.

Download the projects to the corresponding controllers. The implementation of fast data communication via UDP/IP is now completed.

5 Terms and explanations

5.1 Terms

ACR	Attenuation to Crosstalk Radio
ARP	Address Resolution Protocol
AUI	Attachment Unit Interface
BOOTP	Bootstrap Protocol
CSMA/CD	Carrier Sense Multiple Access / Collision Detection
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name Service
DoD	Department of Defense
ELFEXT	Equal Level Far End Cross Talk
FEXT	Far End Cross Talk
FTP	File Transfer Protocol
http	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IEEE	Institute of Electrical and Electronical Engineers
IFG	Interframe Gap
IP	Internet Protocol
LAN	Local Area Network
LLC	Logical Link Control
MAC	Media Access Control Protocol
MDI	Medium Dependent Interface
MDI-X	Medium Dependent Interface, crossed
NEXT	Near End Cross Talk
NIS	Network Information Service
PLS	Physical Layer Signalling
PMA	Physical Medium Attachment
PPP	Point-to-Point Protocol
SMTP	Simple Mail Transfer Protocol
STP	Shielded Twisted Pair
TCP	Transmission Control Protocol
TOS	Type of Service
TP	Twisted Pair
TTL	Time to Live
UDP	User Datagram Protocol
URL	Uniform Resource Locator
UTP	Unshielded Twisted Pair
WAN	Wide Area Network
WWW	World Wide Web

5.2 Explanations

- IP is connectionless
- UDP is connectionless
- TCP is connection oriented (secure communication)
- IP implements the transmission of normal datagrams
- ICMP serves for the transmission of error and information messages
- An IP module of a device can have several IP addresses
- Sockets consist of an IP address and a port and are the interface between TCP and the above application
- Ports are (partly standardized) symbolic numbers
- Transmission of data by an application is performed via an actively open socket
- Reception of data by an application in the target computer is performed by assigning the receive telegram
- (IP address/port) to a passively open socket
- TCP communication phases:
- Connection setup -> data transmission -> closing of connection
- FTP:
  File Transfer Protocol, protocol used for the transfer of files
- HTTP:
  Hyper Text Transfer Protocol, protocol used for the transfer of Internet pages (webservers)
- SMTP:
  Protocol used for sending Emails
- POP3:
  Protocol used for the reception of Emails via an Email server
- Hub:
  Simple "distribution box" for star topology, received telegrams are forwarded to all connected subnetworks
- Switch:
  Intelligent "distribution box" for star topology with knowledge about the structure and the subscribers of the connected subnetworks, specific forwarding of received telegrams to the concerning connected subnetwork or subscriber which reduces the load within the network compared with hubs
- Router, Server:
  Connection of a local network (LAN, e.g. Ethernet) to the "outside world" (WAN, e.g. DSL)
- DHCP:
  Protocol for the automatic IP address assignment performed by the server
- BOOTP:
  Protocol for the automatic IP address assignment performed by the server

6 Index

System Description

AC500

Scalable PLC
for Individual Automation

System Technology of the
PROFIBUS DP Couplers

PROFIBUS

Contents "The PROFIBUS DP Coupler"

1 The PROFIBUS DP Coupler

1.1 Brief overview

1.1.1 Fundamental properties and fields of application

PROFIBUS-DP is designed for the rapid transfer of process data between central controller modules (such as PLC or PC) and decentralized field modules (such as I/O modules, drives and valves) in the field level. The communication occurs mainly cyclic. For intelligent field modules, additionally acyclic communication functions are required for parameter assignment, diagnosis and alarm handling during the running cyclic data transfer.

During normal operation, a central controller (DP master of class 1) cyclically reads the input data of the connected decentralized I/O modules (DP slaves) and sends output data to them. Per slave a maximum of 244 bytes of input and output data can be transferred in one cycle.

Apart from the user data traffic, PROFIBUS-DP provides extensive commissioning and diagnosis functions. The present diagnosis messages of all slave modules are summarized in the master. This enables a quick localization of errors.

Using PROFIBUS-DP, mono-master systems and multi-master systems can be realized. Multi-master systems are built of functionally independent subsystems which each consist of one master and a portion of the slaves which are integrated in the entire system. Normal bus masters cannot exchange information with each other.

PROFIBUS-DP distinguishes two types of masters. The class 1 master carries out the cyclic transfer of user data with the slaves and supplies the user data. The class 1 master can be called by a class 2 master using specific functions. These functions are restricted services, for example the interrogation of diagnosis information of the slaves or the master itself. Thus, a class 2 master is also considered as a programming and diagnosis device.

PROFIBUS-DP uses the hybrid bus access method. This guarantees on the one hand that complex automation devices used as DP masters obtain the opportunity to handle their communication tasks in defined time intervals. On the other hand, it enables the cyclic and real-time related data exchange between the master and peripheral devices (DP slaves). The assigned slave modules on the bus are handled by the master one after the other using the polling operation mode. So, each slave becomes active only after it was requested by the master. This avoids simultaneous access to the bus.

The hybrid access method used with PROFIBUS allows a combined operation of multiple bus masters and even a mixed operation of PROFIBUS-DP and PROFIBUS-FMS in one bus section. This, however, assumes the correct configuration of the bus system and the unique assignment of the slave modules to the masters.

The characteristic properties of a PROFIBUS-DP module are documented in form of an electronic data sheet (modules master data file, GSD file). The modules master data describe the characteristics of a module type completely and clearly in a manufacturer independent format. Using this defined file format strongly simplifies the planning of a PROFIBUS-DP system. Usually the GSD files are provided by the module's manufacturer. In addition, the PROFIBUS user organization (PNO) makes the GSD files of numerous PROFIBUS-DP modules available for a free of charge download via internet in their GSD library. The address of the PROFIBUS user organization (PNO) is: http://www.profibus.com.

1.1.2 Features

Transmission technique:

- RS485, potential separated, insulation voltage up to 850 V.
- Twisted pair cable or optical fibre as a medium for the bus.
- Transfer rate from 9.6 kbit/s up to 12 Mbit/s.
- Bus length up to 1200 m at 9.6 kbit/s and up to 100 m at 12 Mbit/s.
- Up to 32 subscribers (master and slave modules) without repeaters and up to 126 subscribers on one bus with repeaters.
- 9-pole SUB-D socket for bus connection; assignment according to standard.
- Integrated repeater controller.

Communication:

- Up to 244 bytes of input data and 244 bytes of output data per slave, 2944 I/O points max.
- Cyclic user data transfer between DP master and DP slave.
- Acyclic data transfer from master to master.
- Slave configuration check.
- Efficient diagnosis functions, 3 graduated diagnosis messaging levels.
- Synchronization of inputs and/or outputs via control commands.

Protection functions:

- Message transfer with Hamming distance HD = 4.
- Errors during data transfer are detected by the CRC check and cause a repetition of the telegram.
- Access protection for inputs and outputs of the slaves.
- Incorrect parameter settings are avoided since bus subscribers with faulty parameters are not included in the user data operation.
- A failure of a bus subscriber is registered in the master and indicated via a common diagnosis.

Status indication via 4 LEDs

- PWR (green): Supply voltage
- RDY (yellow): Coupler is ready for operation.
- RUN (green): Configuration and communication status.
- STA (yellow): Data exchange.
- ERR (red): PROFIBUS error.

1.2 Technical data

1.2.1 Technical data of the coupler

Coupler type	PROFIBUS coupler
Processor	EC1-160, 48 MHz
Internal power supply with	+5 V, 330 mA typ.
Internal RAM memory (EC1)	256 kbytes
External Flash memory	512 kbytes (firmware)
CE sign	yes

1.2.2 Technical data of the interface

Interface socket	9-pole, SUB-D socket
Transmission standard	EIA RS-485 acc. to EN 50170, potential-free
Transmission protocol	PROFIBUS-DP, 12 Mbaud max.
Transmission rate	Baudrate 9.6 kbit/s up to 12000 kbit/s
Status indication	by 5 LEDs
Number of subscribers (master/slave modules per bus segment)	32 max.
Number of subscribers via repeaters	126 max.

1.3 Connection and data transfer media

1.3.1 Attachment plug for the bus cable

9-pin SUB-D connector, male

Assignment:

Pin No.	Signal	Meaning
1	Shield	Shielding, protective earth
2	not used	
3	RxD/TxD-P	Reception / transmission line, positive
4	CBTR-P	Control signal for repeater (optional)
5	DGND	Reference potential for data lines and +5V
6	VP	+5 V, supply voltage for bus terminating resistors
7	not used	
8	RxD/TxD-N	Reception / transmission line, negative
9	CNTR-N	Control signal for repeater, negative (optional)

Table 1-1: Pin assignment of the attachment plug for the bus cable

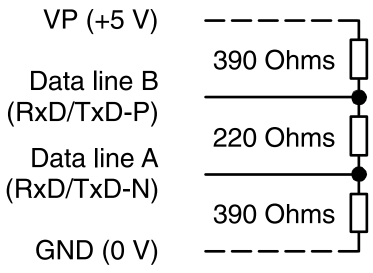
Supplier:

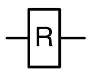
e.g. Erbic® BUS Interface Connector

ERNI Elektroapparate GmbH
Seestraße 9
D-72099 Adelberg, Germany
Phone: +49 7166 50 176
Telefax: +49 7166 50 103
Internet: http://www.erni.com

1.3.2 Bus terminating resistors

The line ends (of the bus segments) have to be terminated using bus terminating resistors according to the drawing below. The bus terminating resistors are usually placed inside the bus connector.





Configuration of the resistors Symbol

Figure: Bus terminating resistors connected to the line ends

1.3.3 Bus cable

Type	Twisted pair cable (shielded)
Wave impedance (cable impedance)	135...165 Ω
Cable capacity (distributed capacitance)	< 30 pF/m
Diameter of line cores (copper)	≥ 0.64 mm
Cross section of line cores	≥ 0.34 mm²
Line resistance per core	≤ 55 Ω/km
Loop resistance (serial resistance of 2 cores)	≤ 110 Ω/km

Supplier:

e.g. UNITRONIC® BUS
U.I. LAPP GmbH
Schulze-Delitzsch-Straße 25
D-70565 Stuttgart, Germany
Phone: (+49) 711 7838 01
Telefax: (+49) 711 7838 264
Internet: http://www.lappkabel.de

1.3.4 Maximum line lengths (bus segment)

1200 m	at a transfer rate of 9.6 / 19.2 / 93.75 kbit/s
1000 m	at a transfer rate of 187.5 kbit/s
400 m	at a transfer rate of 500 kbit/s
200 m	at a transfer rate of 1500 kbit/s
100 m	at a transfer rate of 3000 / 6000 / 12000 kbit/s

Branch lines are generally permissible for baud rates of up to 1500 kbit/s. But in fact they should be avoided for transmission rates higher than 500 kbit/s.

1.3.5 Repeaters

One bus segment can have up to 32 subscribers. Using repeaters a system can be expanded to up to 126 subscribers. Repeaters are also required for longer transfer lines. Please note that a repeater's load to the bus segment is the same as the load of a normal bus subscriber. The sum of normal bus subscribers and repeaters in one bus segment must not exceed 32.

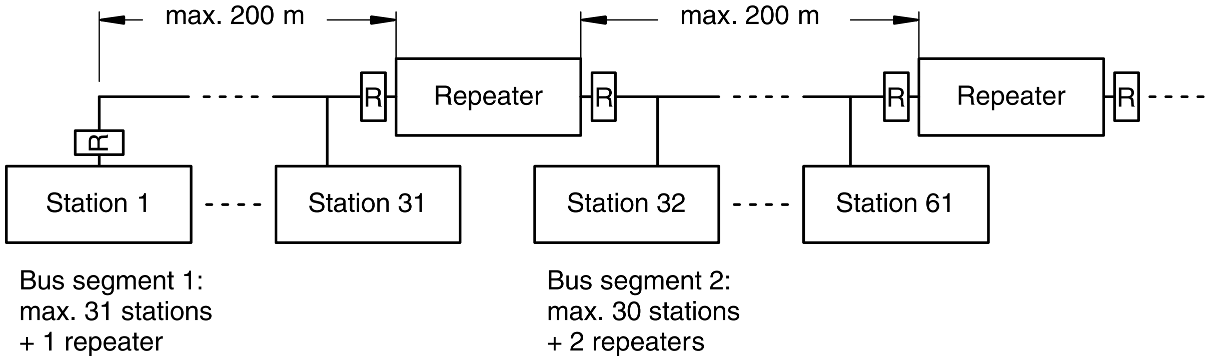


Figure: Principle example for a PROFIBUS-DP system with repeaters (1500 kbit/s baud rate)

1.4 Possibilities for networking

The PROFIBUS coupler is connected to the bus via the 9-pole SUB-D socket. For EMC suppression and protection against dangerous contact voltages, the shield of the bus line has to be connected to protective earth outside the housing.

1.4.1 Single master system

The single master system is the simplest version of a PROFIBUS network. It consists of a class 1 DP master and one or more DP slaves. Up to 31 DP slaves can be connected to the bus without using a repeater. If the number of bus segments is increased by means of repeaters, up to 126 DP slaves can be handled. The line ends of the bus segments have to be terminated using bus terminating resistors.

The DP master of class 1 is able to:

1. Parameterize DP slaves (e.g. timing supervision, bus interchange).
2. Configure DP slaves (e.g. type / number of channels).
3. Read input and output data of the DP slaves.
4. Write output data of the DP slaves.
5. Read diagnosis data of the DP slaves.
6. Send control commands to the DP slaves (e.g. freezing input signals).

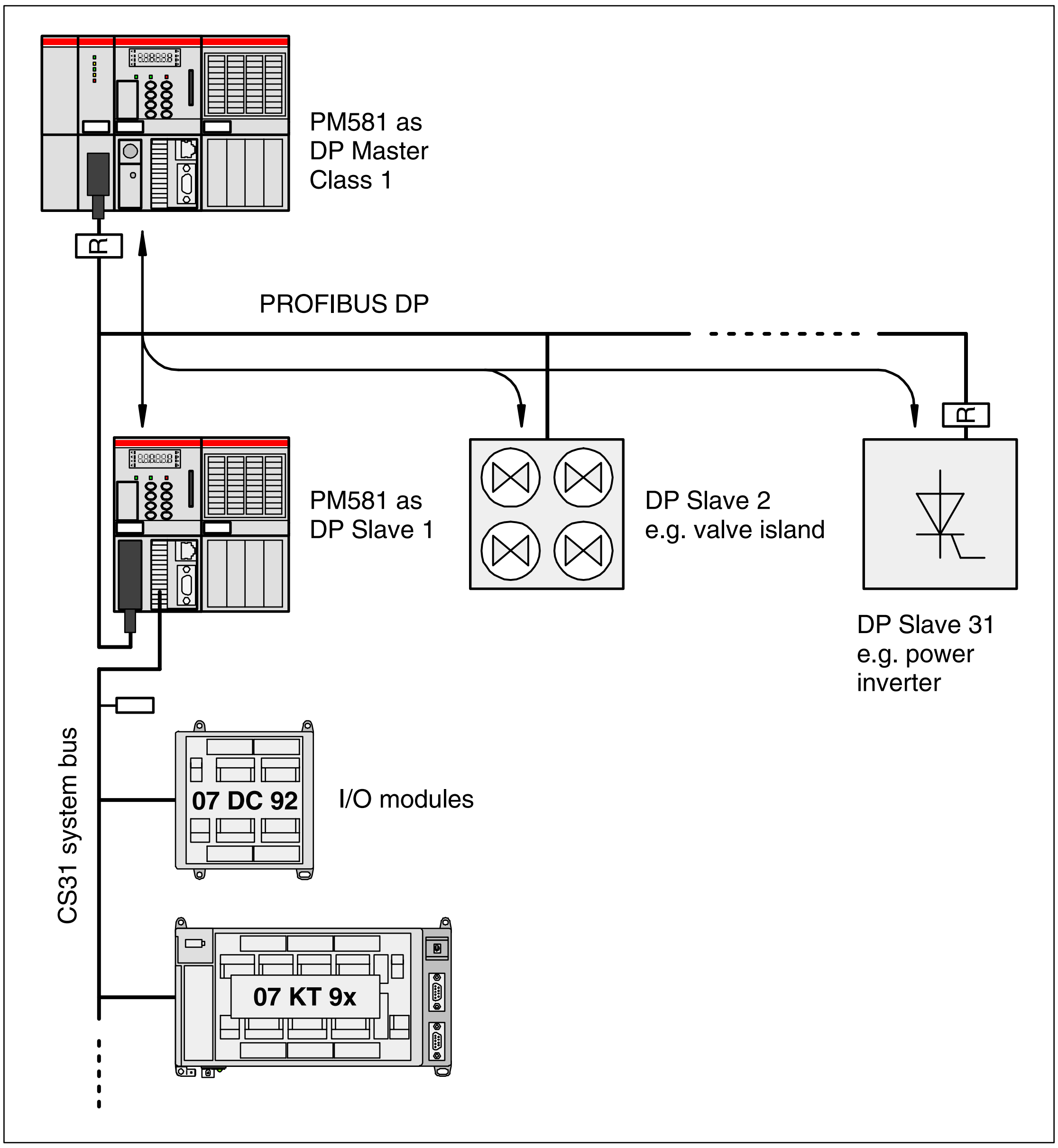


Figure: Single master system example

1.4.2 Multi master system

A PROFIBUS network containing several DP masters is called a multi-master system. Up to 32 subscribers (DP masters and DP slaves) can be operated on one bus segment. Using repeaters the system can be expanded to up to 126 subscribers. In a multi-master system no data exchange between the DP masters is performed. The entire system is divided into logical subsystems inside of which one DP master communicates with the assigned DP slaves. Each DP slave can be assigned to only one DP master. The master has unlimited access to its assigned slaves while all other masters on the bus can only read the input and output data of these slaves.

All DP masters of class 1 (normal bus master, here: AC500) and class 2 (commissioning device, typically a PC) can read the input and output data of all slaves.

Additionally the DP masters of class 1 and class 2 have the following access possibilities to their assigned DP slaves. They are able to:

- Parameterize DP slaves (e.g. timing supervision, bus interchange).
- Configure DP slaves (e.g. type / number of channels).
- Write output data of the DP slaves.
- Read diagnosis data of the DP slaves.
- Send control commands to the DP slaves (e.g. freezing input signals).

A DP master of class 2 is additionally able to:

- Read and write configuration data of the class 1 DP masters.
- Read configuration data of the DP slaves.
- Read diagnosis data of the class 1 DP masters.
- Read out the diagnosis data of the DP slaves assigned to the respective DP master.

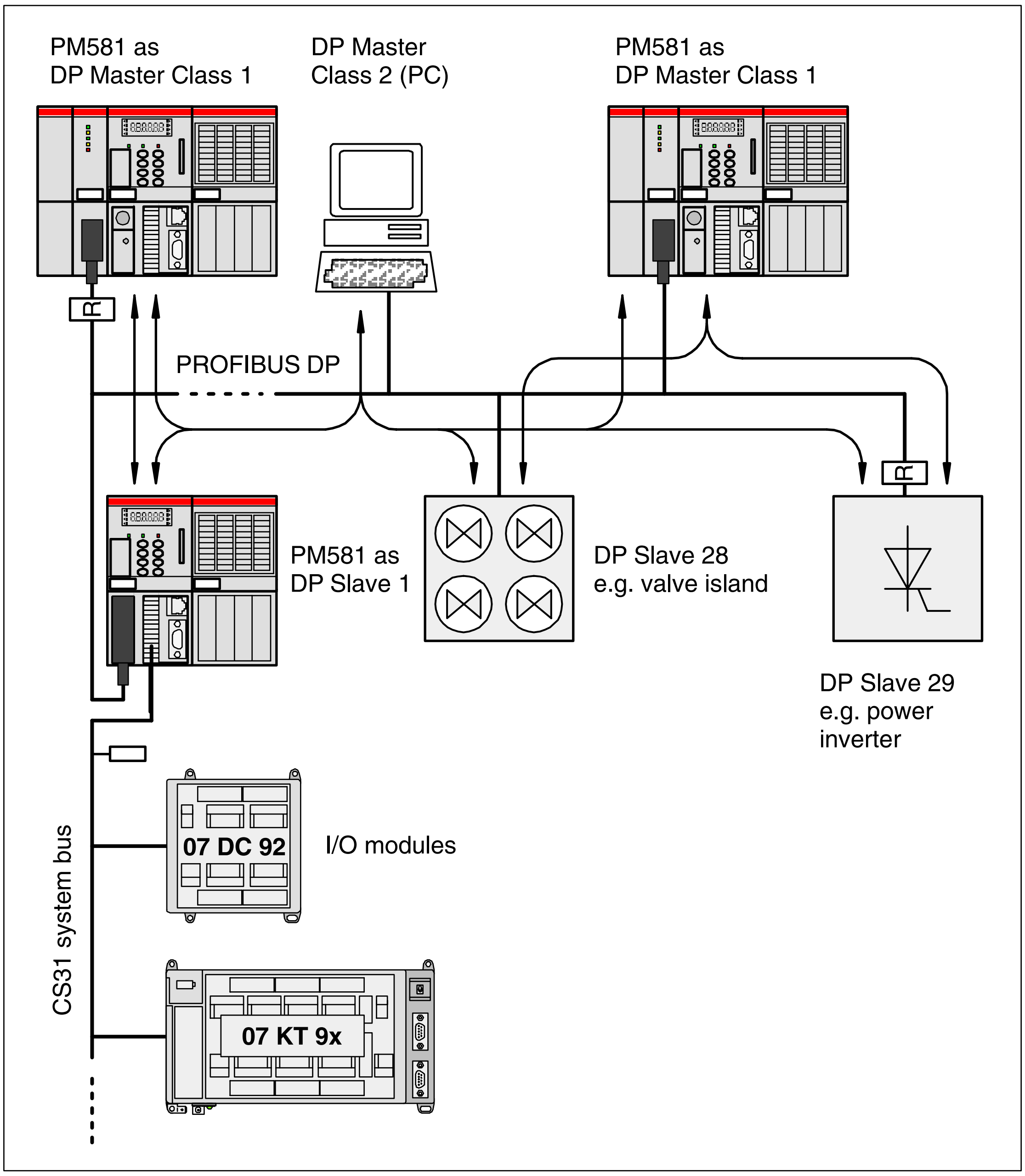


Figure: Multi master system example

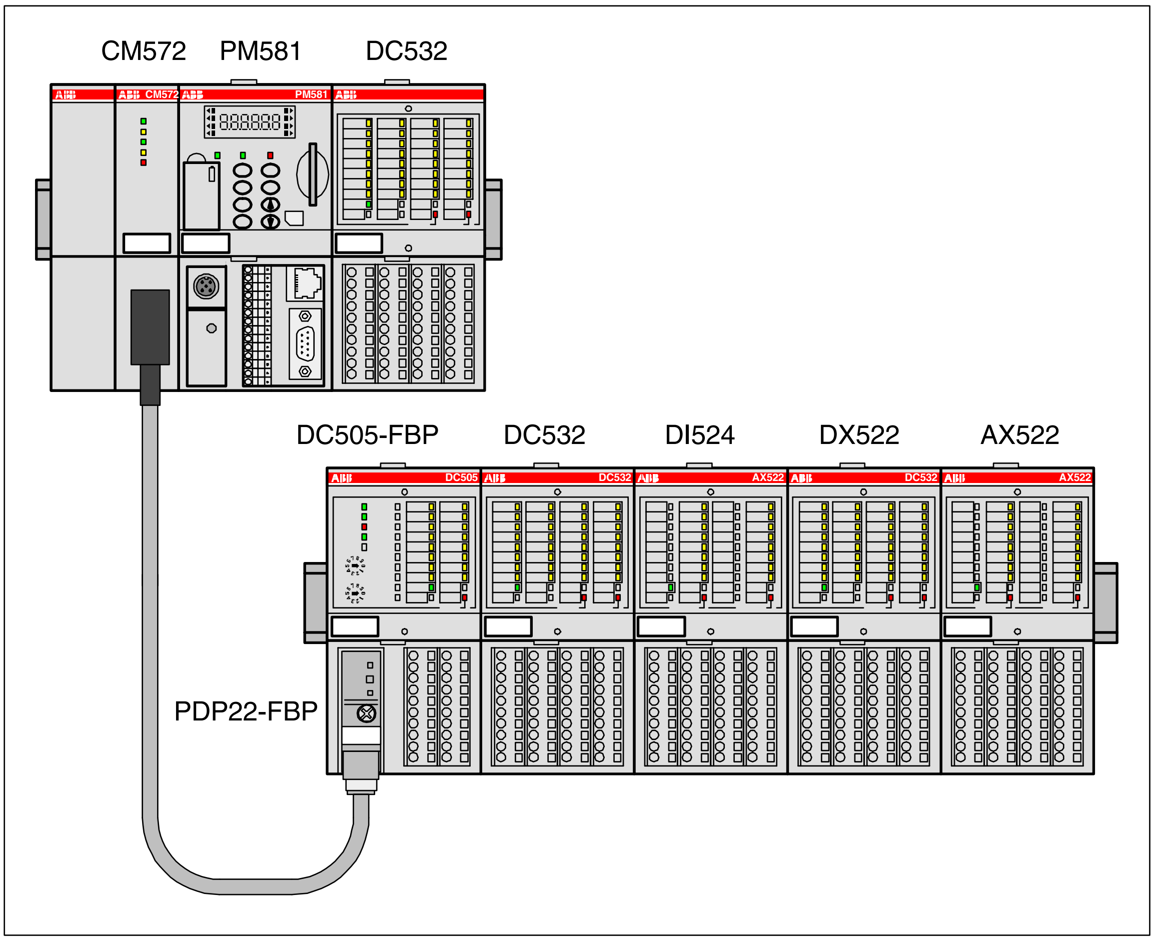
1.5 PROFIBUS DP configuration example

1.5.1 Configuration

The integration of the coupler in the PLC configuration is an assumption for the correct function of the PROFIBUS coupler CM572. Configuration of the coupler and the connected PROFIBUS subscribers is done using the tool SYCON.net which is part of the CoDeSys programming software.

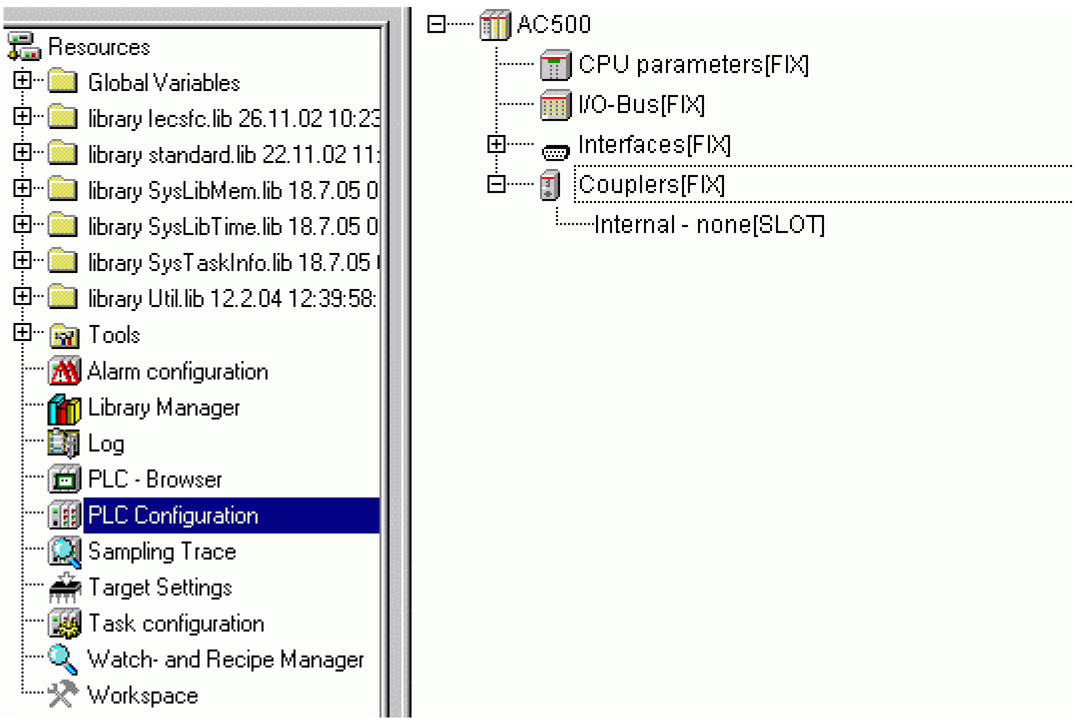
The PROFIBUS DP coupler CM572 has to be configured as the PROFIBUS master. The PROFIBUS slave functionality is made available via the FBP interface and an FBP PROFIBUS plug.

The following construction is used as an example of a PROFIBUS configuration:



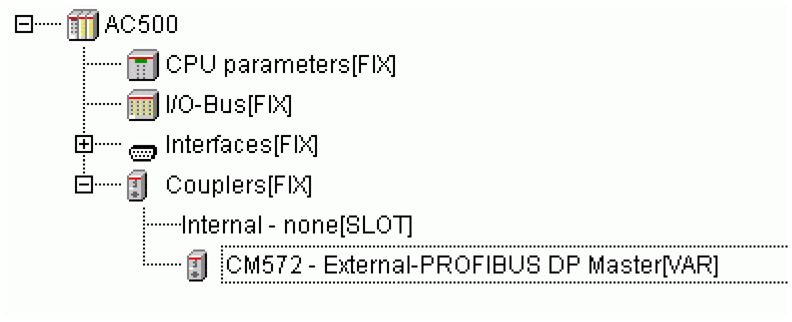
AC500 configuration: CPU (PM581) with an expansion DC532 at the I/O-Bus as well as one decentralized expansion over PROFIBUS (CM572) with the devices PDP22-FBP, DC505-FBP as well as the S500 modules DC532, DI524, DX522 and AX522.

PLC configuration

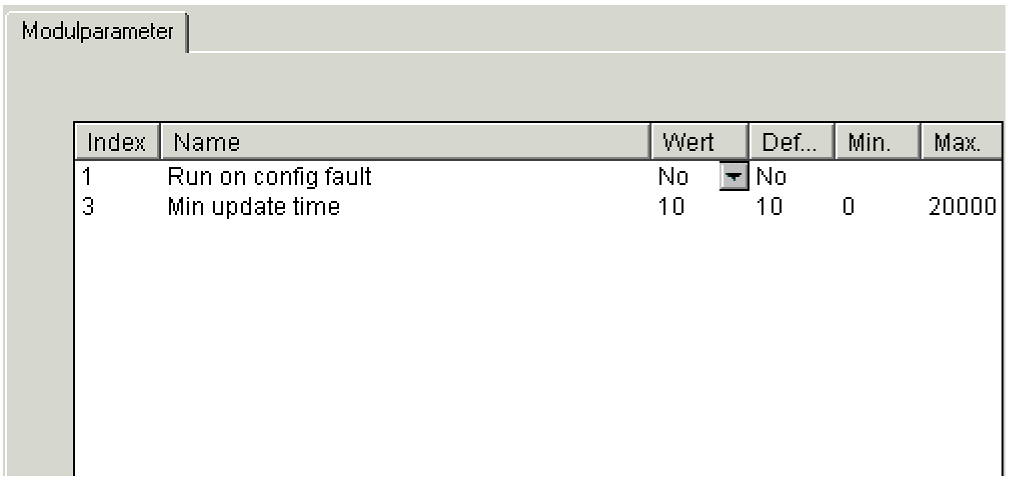

Resources
Global Variables
library Iecsfc.lib 26.11.02 10:23
library standard.lib 22.11.02 11:
library SysLibMem.lib 18.7.05 0
library SysLibTime.lib 18.7.05 0
library SysTaskInfo.lib 18.7.05
library Util.lib 12.2.04 12:39:58:
Tools
Alarm configuration
Library Manager
Log
PLC - Browser
PLC Configuration
Sampling Trace
Target Settings
Task configuration
Watch- and Recipe Manager
Workspace

AC500
CPU parameters[FIX]
I/O-Bus[FIX]
Interfaces[FIX]
Couplers[FIX]
Internal - none[SLOT]

To add the coupler: Select "Couplers", press the right mouse button and then select "Append Subelement" -> "CM572".


AC500
CPU parameters[FIX]
I/O-Bus[FIX]
Interfaces[FIX]
Couplers[FIX]
Internal - none[SLOT]
CM572 - External-PROFIBUS DP Master[VAR]

Do not change the default values for the coupler parameters.

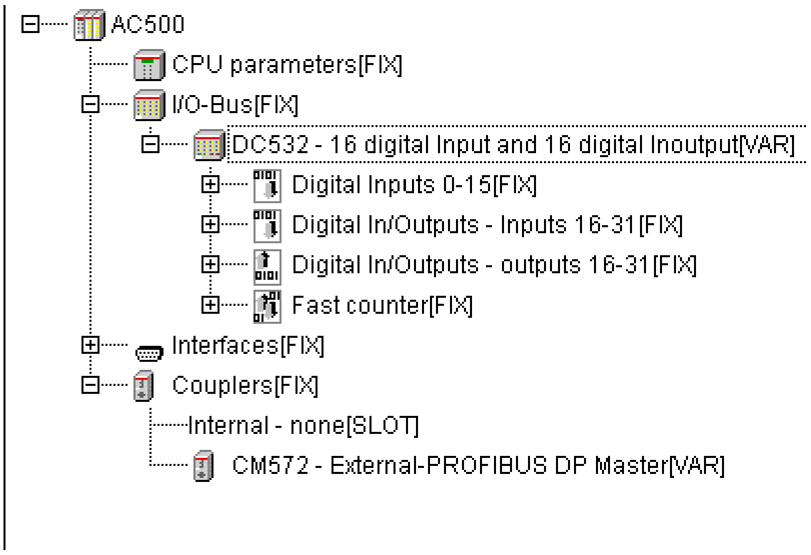


Modulparameter

Index	Name	Wert	Def...	Min.	Max.
1	Run on config fault	No	No		
3	Min update time	10	10	0	20000

The coupler is now integrated in the PLC configuration.

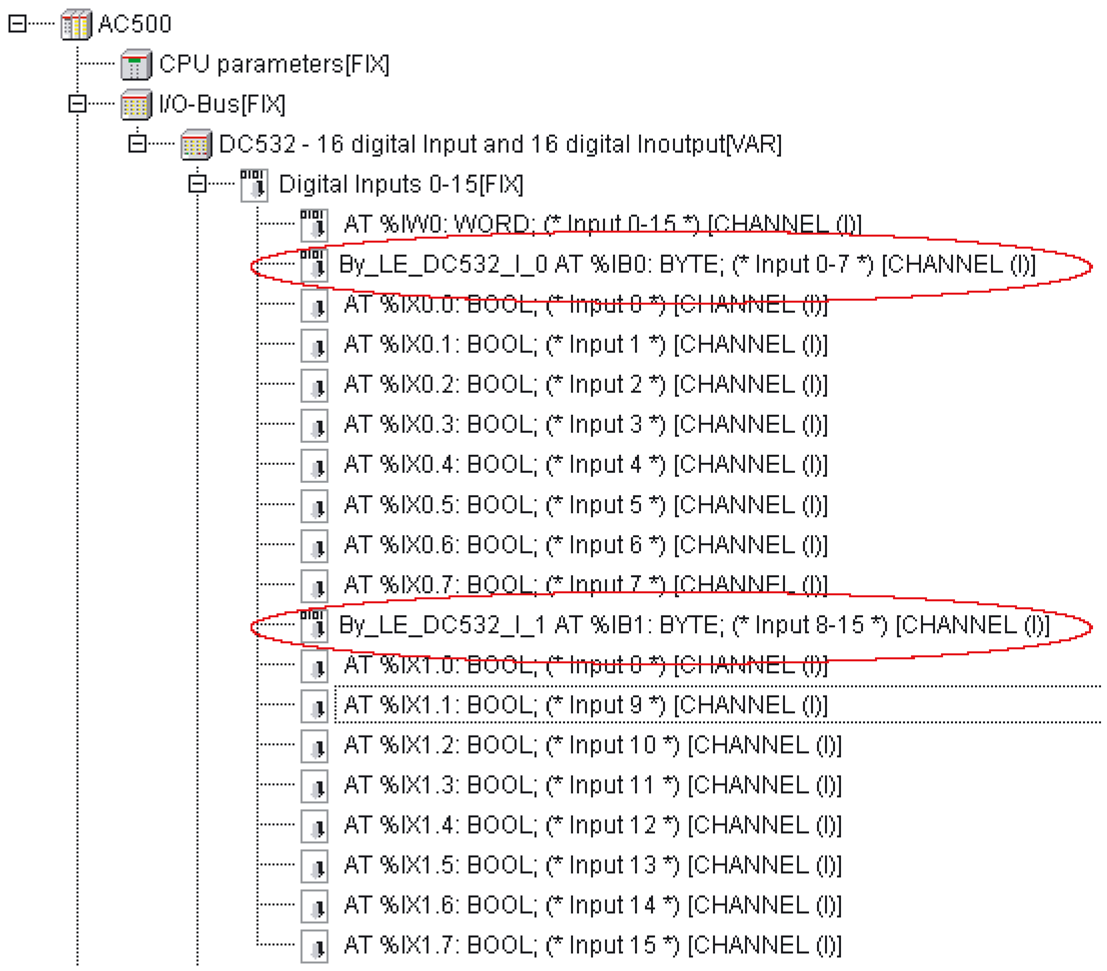
For the following example, an I/O module (DC532) is then inserted into this configuration at the I/O-Bus.



```
AC500
  CPU parameters[FIX]
  I/O-Bus[FIX]
    DC532 - 16 digital Input and 16 digital Inoutput[VAR]
      Digital Inputs 0-15[FIX]
      Digital In/Outputs - Inputs 16-31[FIX]
      Digital In/Outputs - outputs 16-31[FIX]
      Fast counter[FIX]
  Interfaces[FIX]
  Couplers[FIX]
    Internal - none[SLOT]
    CM572 - External-PROFIBUS DP Master[VAR]
```

For the module settings of the DC532, the default values are left as is.

Declaration of the I/Os of the devices is done byte-by-byte.

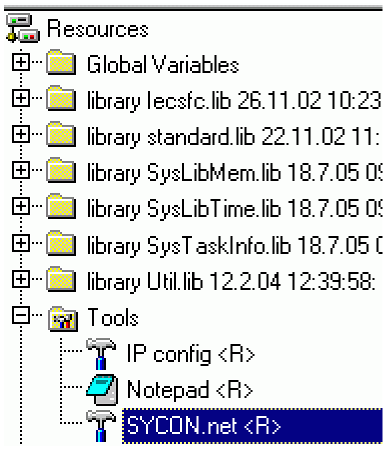


```
AC500
  CPU parameters[FIX]
  I/O-Bus[FIX]
    DC532 - 16 digital Input and 16 digital Inoutput[VAR]
      Digital Inputs 0-15[FIX]
        AT %IW0: WORD; (* Input 0-15 *) [CHANNEL (I)]
        By_LE_DC532_I_0 AT %IB0: BYTE; (* Input 0-7 *) [CHANNEL (I)]
        AT %IX0.0: BOOL; (* Input 0 *) [CHANNEL (I)]
        AT %IX0.1: BOOL; (* Input 1 *) [CHANNEL (I)]
        AT %IX0.2: BOOL; (* Input 2 *) [CHANNEL (I)]
        AT %IX0.3: BOOL; (* Input 3 *) [CHANNEL (I)]
        AT %IX0.4: BOOL; (* Input 4 *) [CHANNEL (I)]
        AT %IX0.5: BOOL; (* Input 5 *) [CHANNEL (I)]
        AT %IX0.6: BOOL; (* Input 6 *) [CHANNEL (I)]
        AT %IX0.7: BOOL; (* Input 7 *) [CHANNEL (I)]
        By_LE_DC532_I_1 AT %IB1: BYTE; (* Input 8-15 *) [CHANNEL (I)]
        AT %IX1.0: BOOL; (* Input 8 *) [CHANNEL (I)]
        AT %IX1.1: BOOL; (* Input 9 *) [CHANNEL (I)]
        AT %IX1.2: BOOL; (* Input 10 *) [CHANNEL (I)]
        AT %IX1.3: BOOL; (* Input 11 *) [CHANNEL (I)]
        AT %IX1.4: BOOL; (* Input 12 *) [CHANNEL (I)]
        AT %IX1.5: BOOL; (* Input 13 *) [CHANNEL (I)]
        AT %IX1.6: BOOL; (* Input 14 *) [CHANNEL (I)]
        AT %IX1.7: BOOL; (* Input 15 *) [CHANNEL (I)]
```

Inputs:	By_LE_DC532_I_0 By_LE_DC532_I_1 By_LE_DC532_I_2 By_LE_DC532_I_3
Outputs:	By_LE_DC532_O_0 By_LE_DC532_O_1

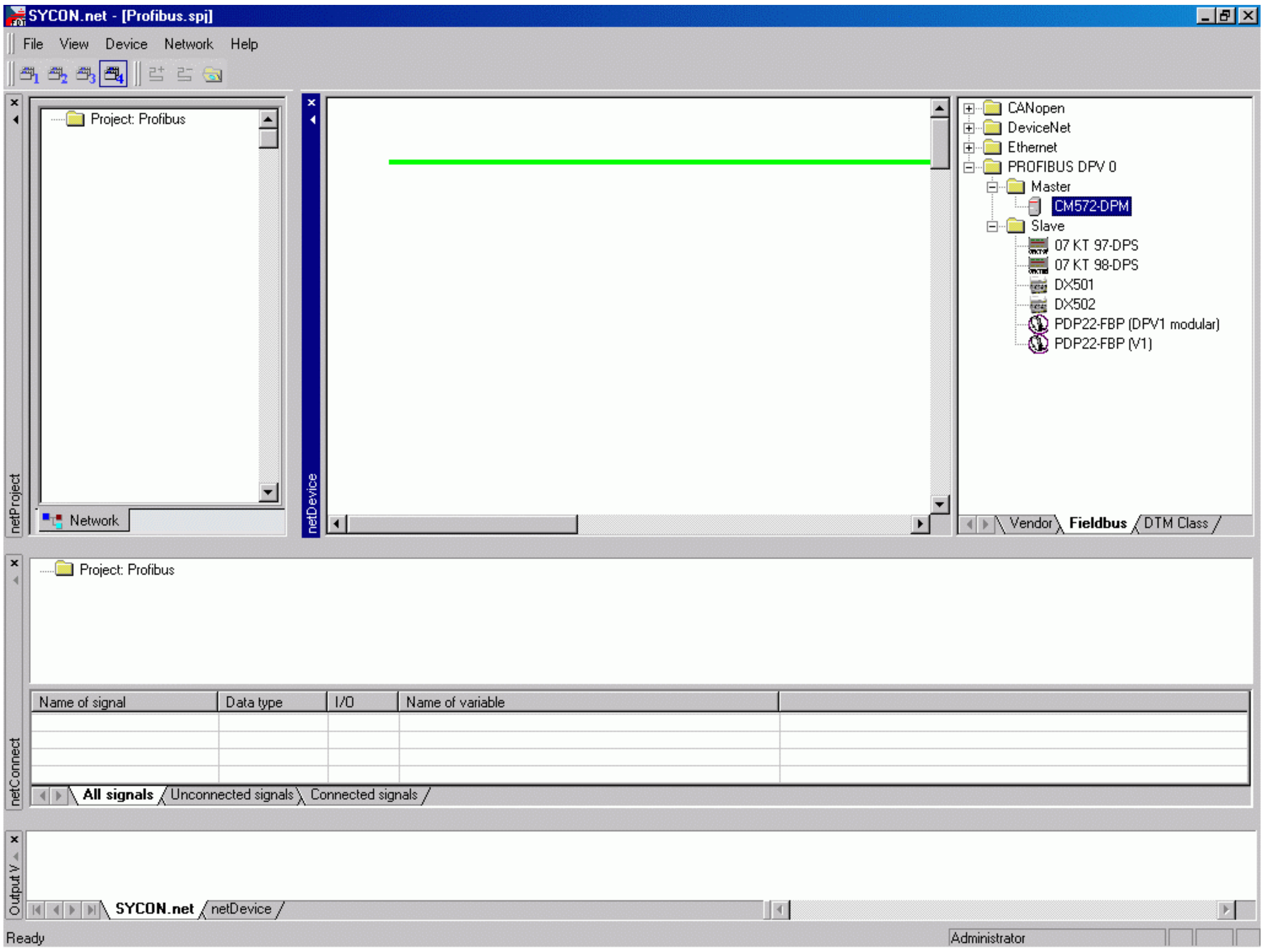
Configuration using SYCON.net

When configuring the PROFIBUS coupler, the configuration data are a definite element of a project. They are created in CoDeSys using the tool SYCON.net (Resources tab -> Tools / SYCON.net). The transfer of the configuration data to the coupler is done within the SYCON.netTool.

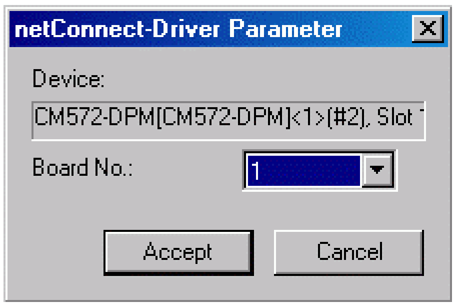


```
Resources
  Global Variables
  library Iecsfc.lib 26.11.02 10:23
  library standard.lib 22.11.02 11:
  library SysLibMem.lib 18.7.05 0
  library SysLibTime.lib 18.7.05 0
  library SysTaskInfo.lib 18.7.05 (
  library Util.lib 12.2.04 12:39:58:
  Tools
    IP config <R>
    Notepad <R>
    SYCON.net <R>
```

The following overview is loaded after starting the configuration tool SYCON.net.



In the top right window, click on the entry "CM572-DPM" and drag it onto the green line displayed in the middle window. Correct insertion positions are displayed by a "+".


netConnect-Driver Parameter

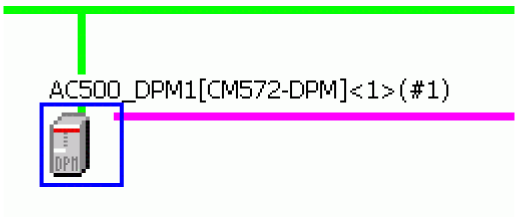
Device:

CM572-DPM[CM572-DPM]<1>(#2), Slot

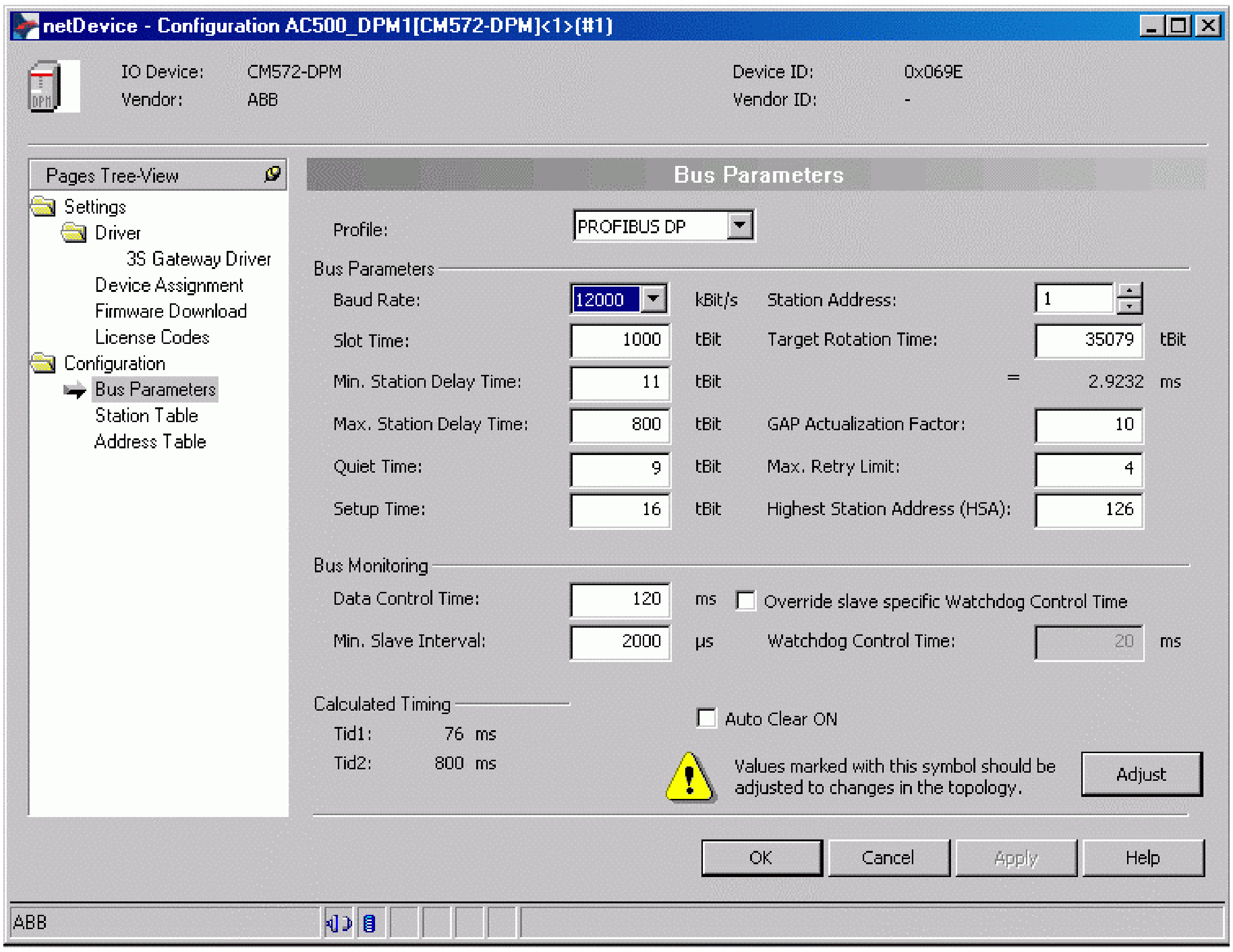
Board No.: 1

Accept | Cancel

A dialog appears where you have to select the card number according to the coupler slot. The first slot left of the CPU is slot 1 (or card number 1).



In order to configure the coupler, place the cursor on the "CM572" icon and then press the right mouse button and select "Configuration".


netDevice - Configuration AC500_DPM1[CM572-DPM]<1>(#1)

IO Device: CM572-DPM
Vendor: ABB
Device ID: 0x069E
Vendor ID: -

Pages Tree-View
- Settings
  - Driver
    - 3S Gateway Driver
  - Device Assignment
  - Firmware Download
  - License Codes
- Configuration
  - Bus Parameters
  - Station Table
  - Address Table

Bus Parameters

Profile: PROFIBUS DP

Bus Parameters

Baud Rate:	12000	kBit/s	Station Address:	1	
Slot Time:	1000	tBit	Target Rotation Time:	35079	tBit
Min. Station Delay Time:	11	tBit	=	2.9232	ms
Max. Station Delay Time:	800	tBit	GAP Actualization Factor:	10	
Quiet Time:	9	tBit	Max. Retry Limit:	4	
Setup Time:	16	tBit	Highest Station Address (HSA):	126	

Bus Monitoring

Data Control Time:	120	ms	☐ Override slave specific Watchdog Control Time		
Min. Slave Interval:	2000	µs	Watchdog Control Time:	20	ms

Calculated Timing

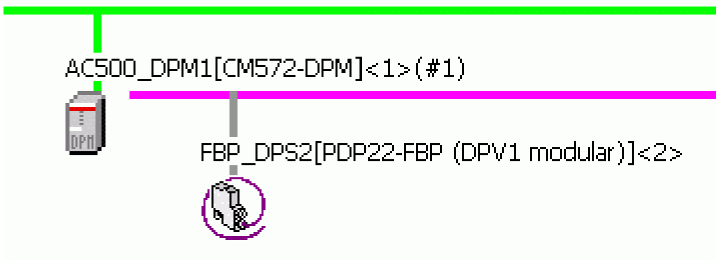
Tid1: 76 ms
Tid2: 800 ms

☐ Auto Clear ON

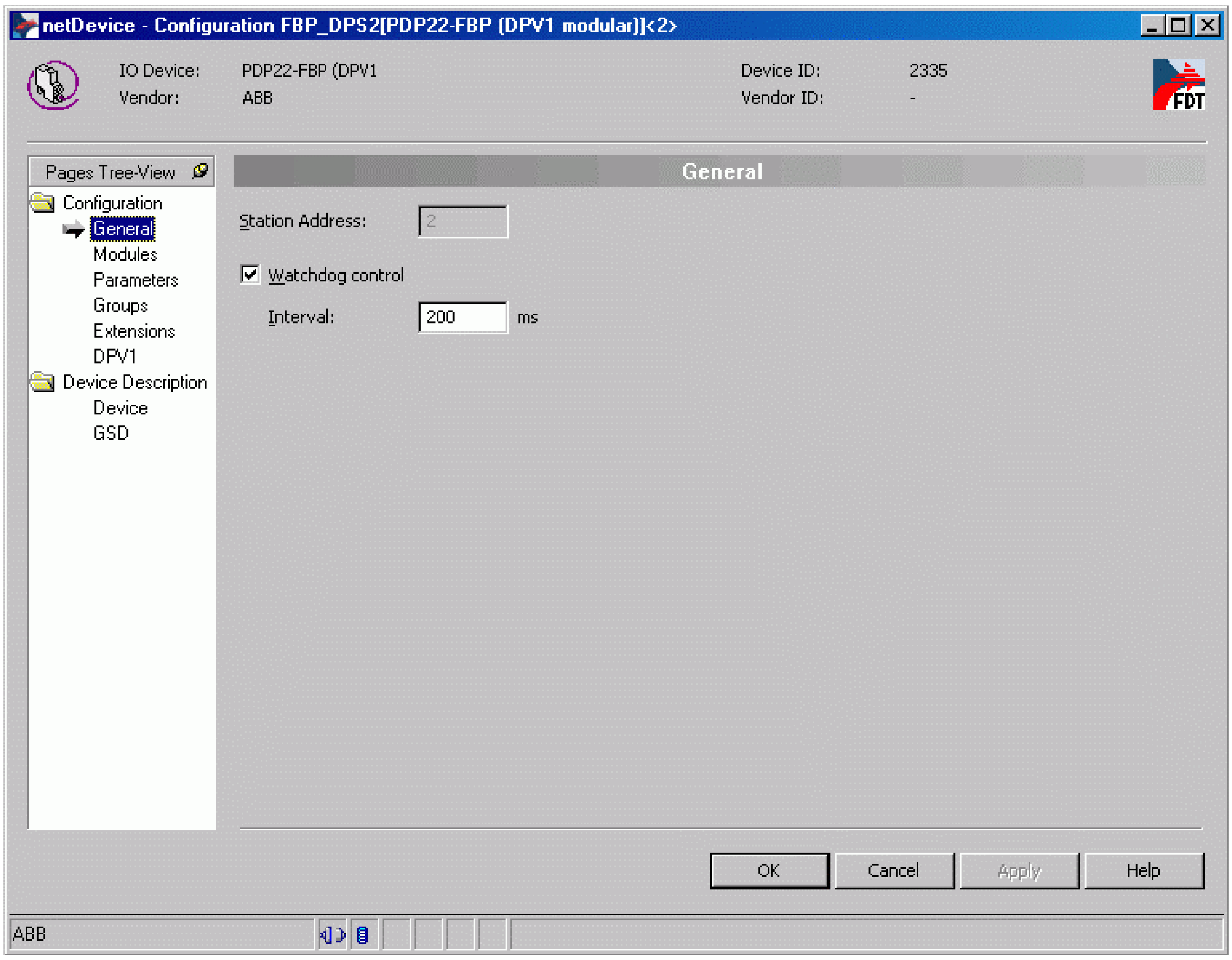
Values marked with this symbol should be adjusted to changes in the topology.

In the folder "Bus parameters", choose the baud rate for the PROFIBUS and then confirm the remaining parameters with "OK".

Next the PROFIBUS slaves are configured. In the top right window, click on the entry "PDP22-FBP (DPV1 modular)" in the folder "PROFIBUS DPV 0/Slave" and drag it onto the purple line (PROFIBUS line) displayed in the middle window. Correct insertion positions are displayed by a "+".


AC500_DPM1[CM572-DPM]<1>(#1)
DPM
FBP_DPS2[PDP22-FBP (DPV1 modular)]<2>

In order to configure the PROFIBUS slave, select the slave, then press the right mouse button and choose "Configuration".


netDevice - Configuration FBP_DPS2[PDP22-FBP (DPV1 modular)]<2>

IO Device: PDP22-FBP (DPV1
Vendor: ABB
Device ID: 2335
Vendor ID: -

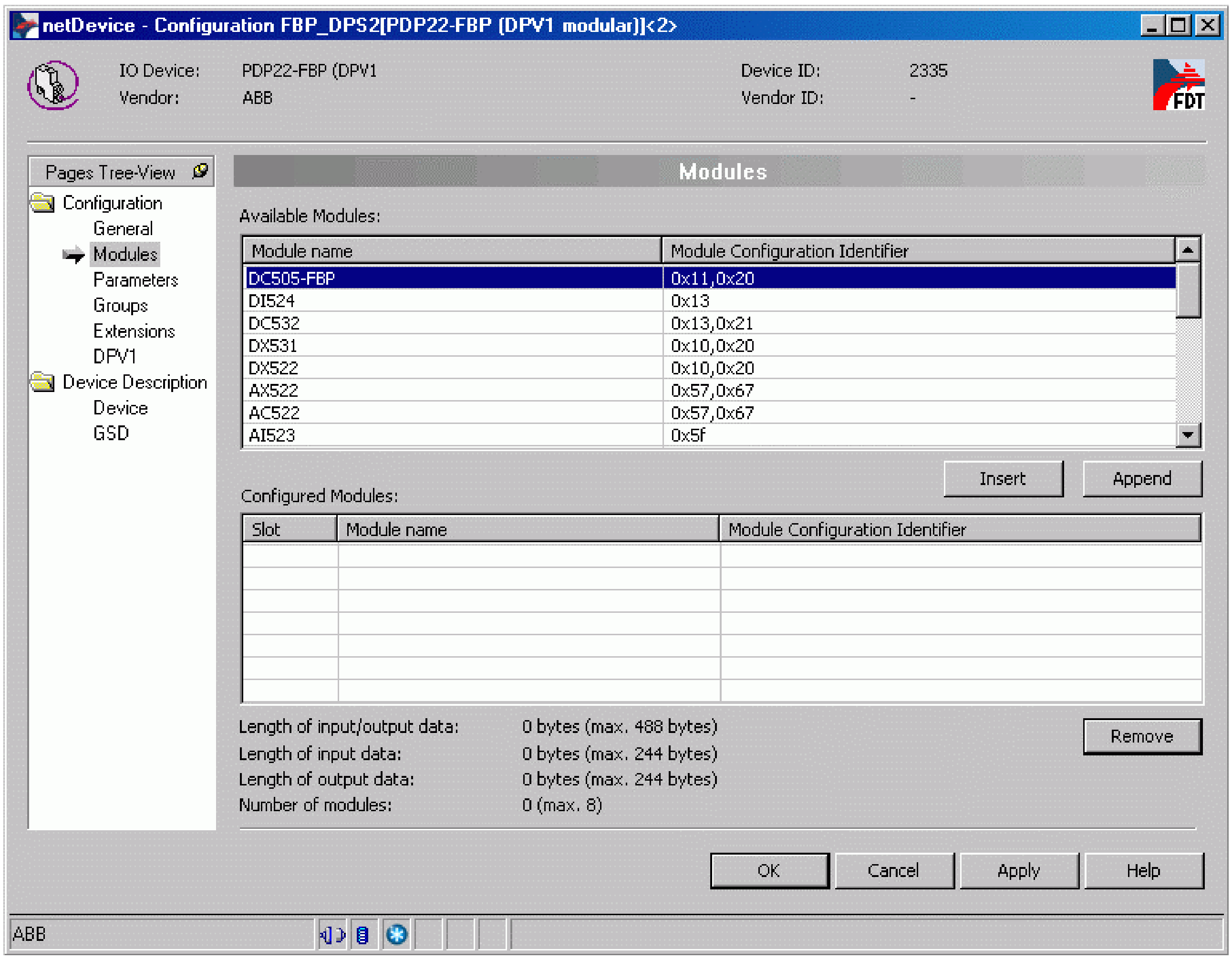
Station Address: 2

[✓] Watchdog control

Interval: 200 ms

In this example, the folder "General" remains unchanged. If the station address should not equal the HW address, this can be adapted correspondingly after the configuration of the slaves in the configuration window of the master in the folder "station address".

In the next step, all the modules are configured, which are connected to the PDP22-FBP DPV1. For this, change into the folder "Configuration/Modules".



netDevice - Configuration FBP_DPS2[PDP22-FBP (DPV1 modular)]<2>

IO Device: PDP22-FBP (DPV1   Device ID: 2335
Vendor: ABB   Vendor ID: -

Pages Tree-View
- Configuration
  - General
  - Modules
  - Parameters
  - Groups
  - Extensions
  - DPV1
- Device Description
  - Device
  - GSD

Modules

Available Modules:

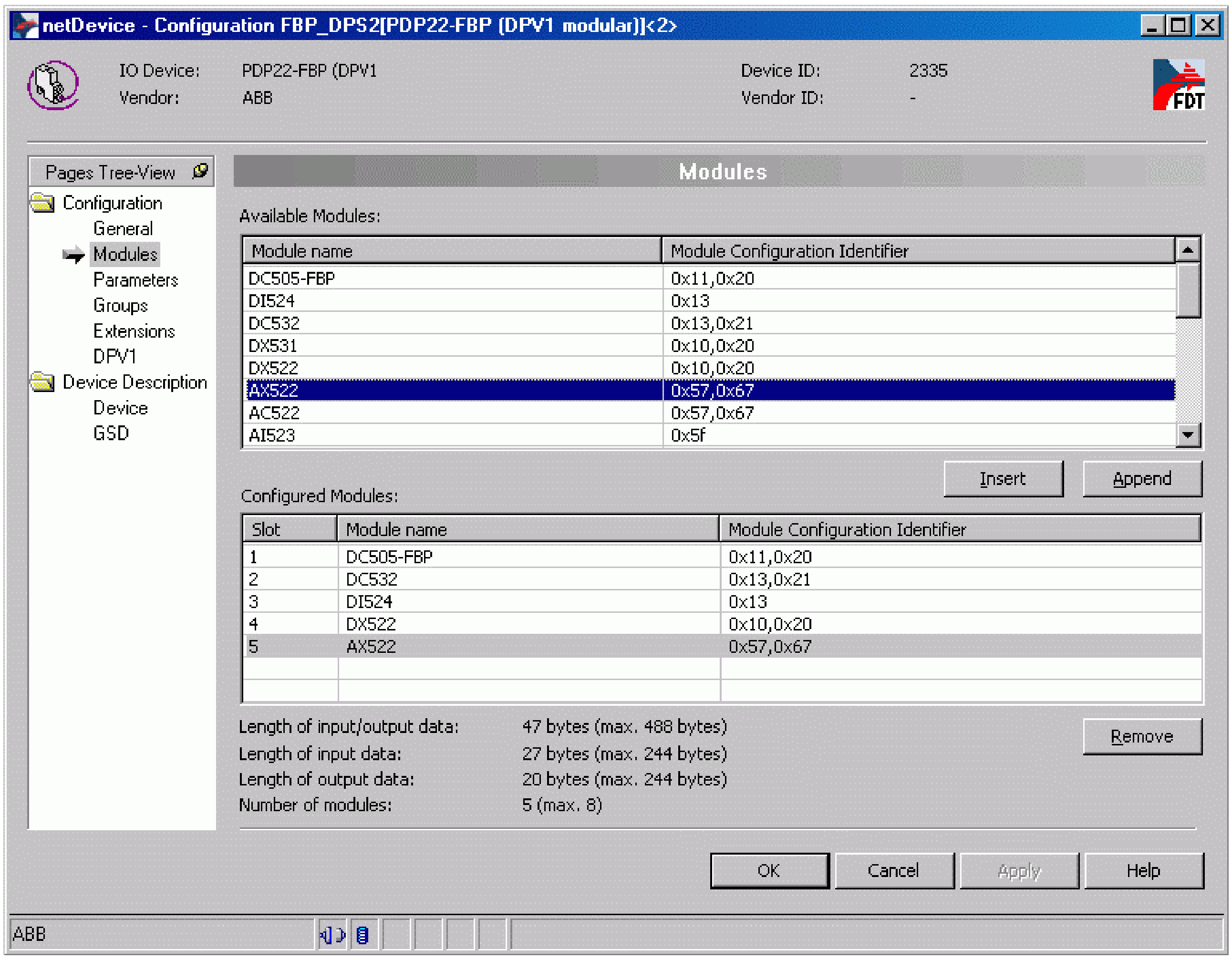
Module name	Module Configuration Identifier
DC505-FBP	0x11,0x20
DI524	0x13
DC532	0x13,0x21
DX531	0x10,0x20
DX522	0x10,0x20
AX522	0x57,0x67
AC522	0x57,0x67
AI523	0x5f

Configured Modules:

Slot	Module name	Module Configuration Identifier

Length of input/output data:	0 bytes (max. 488 bytes)
Length of input data:	0 bytes (max. 244 bytes)
Length of output data:	0 bytes (max. 244 bytes)
Number of modules:	0 (max. 8)

Select the individual modules in the window "Available Modules" beginning with the S500 head station "DC505-FBP" and append them to the "configured modules" by using "Append". The order of the "configured modules" should represent the existing hardware.


netDevice - Configuration FBP_DPS2[PDP22-FBP (DPV1 modular)]<2>

IO Device: PDP22-FBP (DPV1
Vendor: ABB
Device ID: 2335
Vendor ID: -

Available Modules:

Module name	Module Configuration Identifier
DC505-FBP	0x11,0x20
DI524	0x13
DC532	0x13,0x21
DX531	0x10,0x20
DX522	0x10,0x20
AX522	0x57,0x67
AC522	0x57,0x67
AI523	0x5f

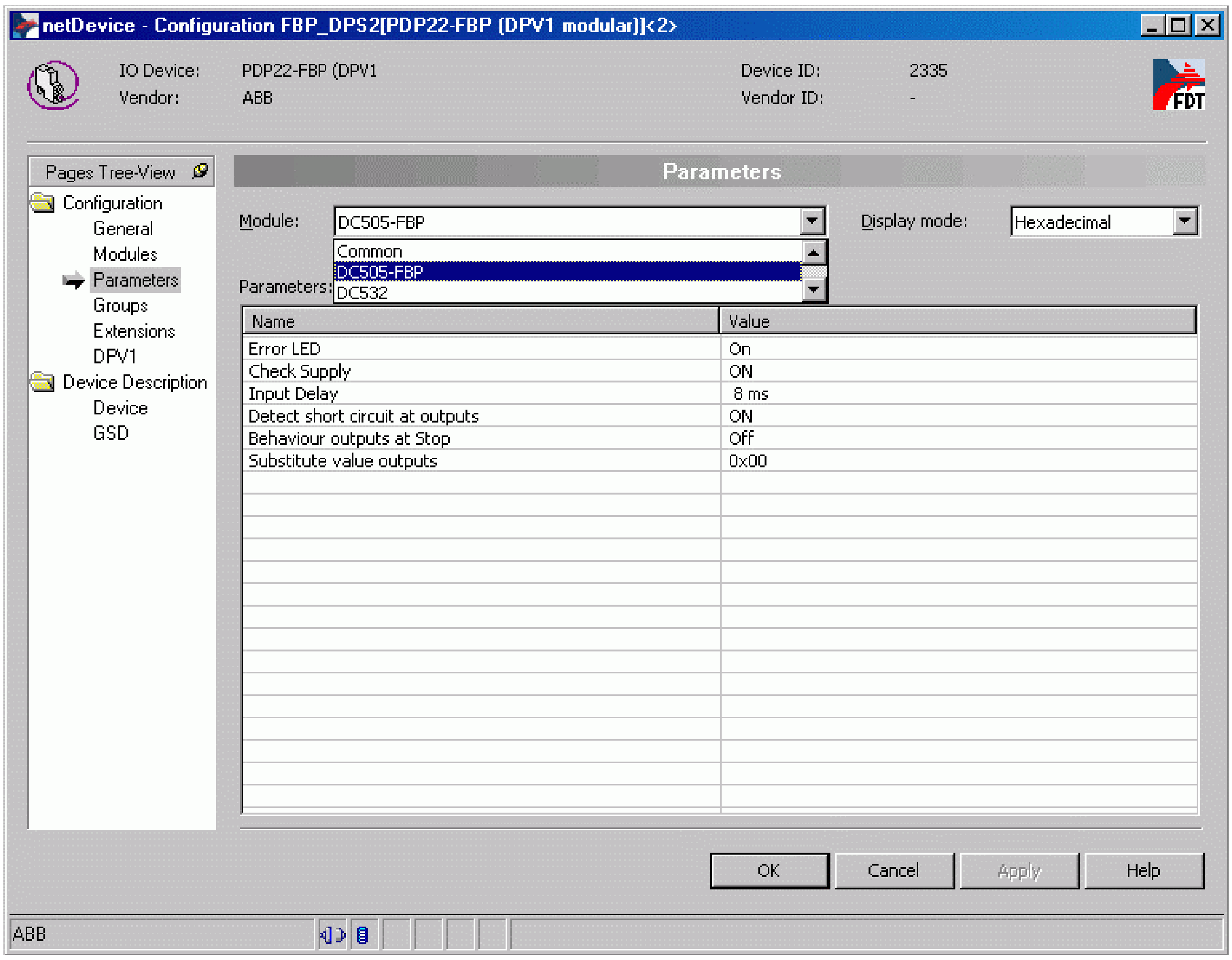
Configured Modules:

Slot	Module name	Module Configuration Identifier
1	DC505-FBP	0x11,0x20
2	DC532	0x13,0x21
3	DI524	0x13
4	DX522	0x10,0x20
5	AX522	0x57,0x67

Length of input/output data:	47 bytes (max. 488 bytes)
Length of input data:	27 bytes (max. 244 bytes)
Length of output data:	20 bytes (max. 244 bytes)
Number of modules:	5 (max. 8)

Now the modules are configured.

Next the parameters of the individual modules are defined. Change to the folder "Configuration/Parameters" and select the first device "DC505-FBP" using "Module".



IO Device: PDP22-FBP (DPV1
Vendor: ABB
Device ID: 2335
Vendor ID: -

Module: DC505-FBP
Display mode: Hexadecimal

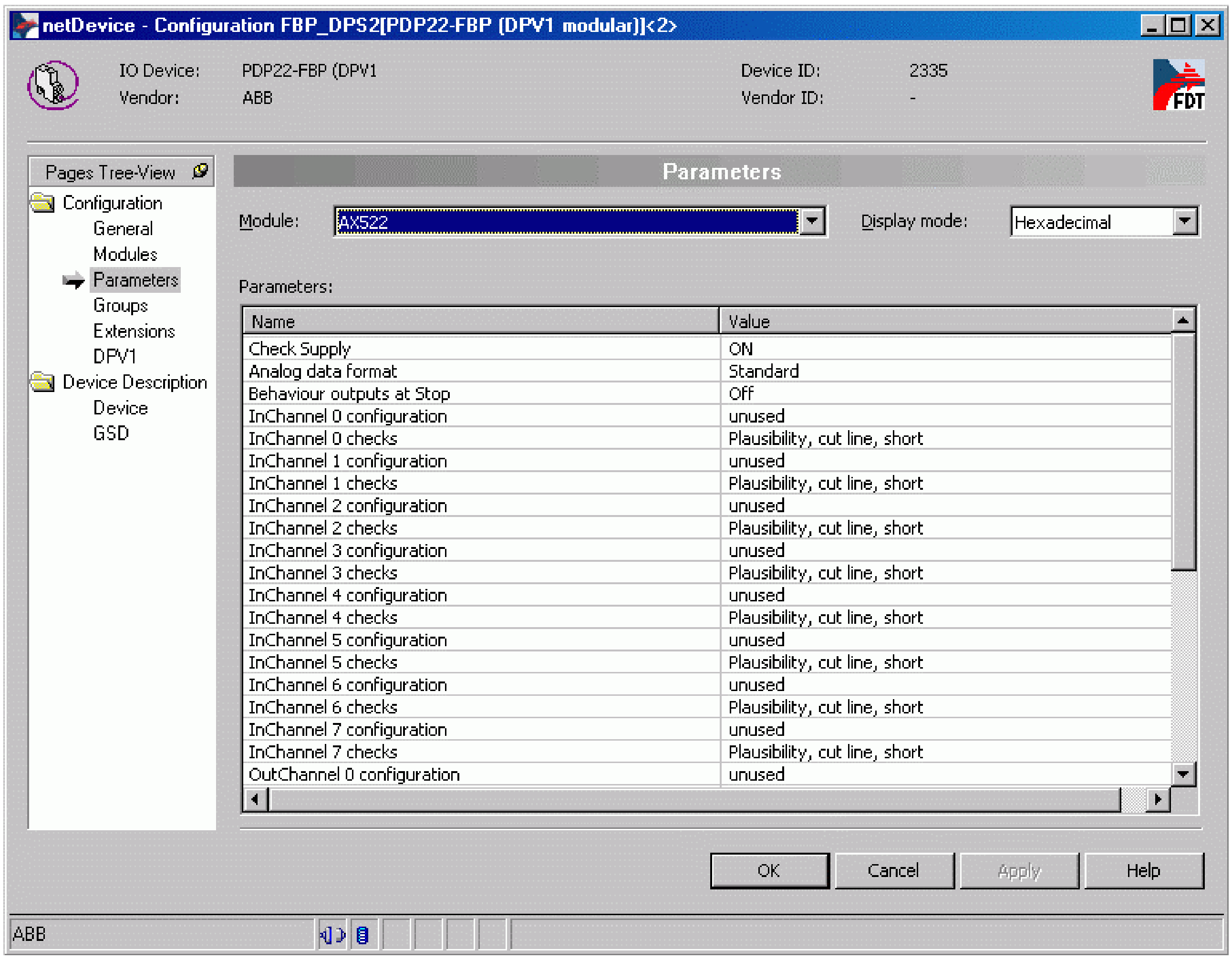
Parameters: Common / DC505-FBP / DC532

Name	Value
Error LED	On
Check Supply	ON
Input Delay	8 ms
Detect short circuit at outputs	ON
Behaviour outputs at Stop	Off
Substitute value outputs	0x00

In this tab, you can set the device parameters of the individual I/O modules.

A selection of possible parameters can be displayed by double clicking on the settings in the "Value" column.

For the digital devices the settings always apply to the entire module. For the analog devices the settings additionally have to be performed for each channel.



netDevice - Configuration FBP_DPS2[PDP22-FBP (DPV1 modular)]<2>

IO Device: PDP22-FBP (DPV1
Vendor: ABB
Device ID: 2335
Vendor ID: -

Parameters

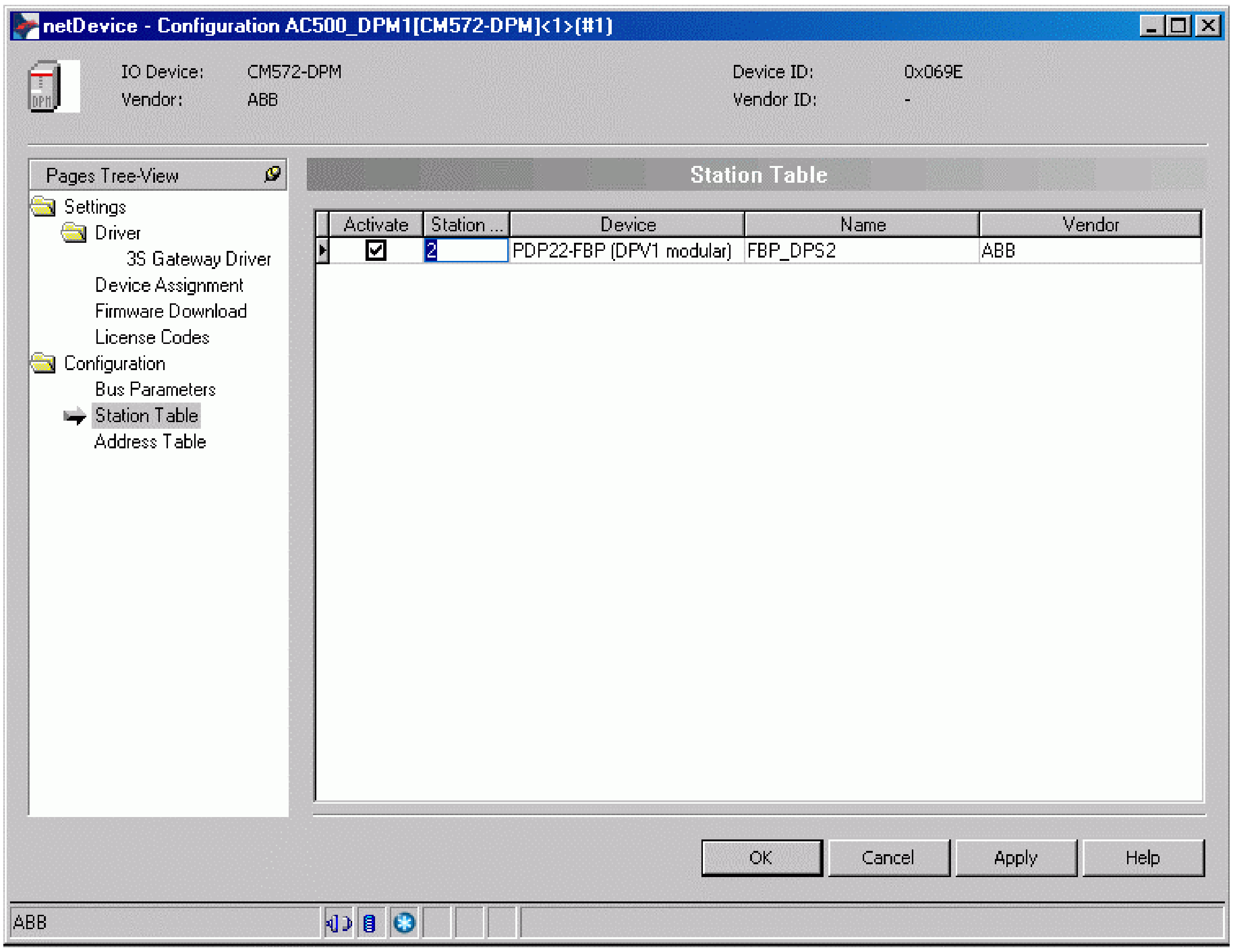
Module: AX522
Display mode: Hexadecimal

Parameters:

Name	Value
Check Supply	ON
Analog data format	Standard
Behaviour outputs at Stop	Off
InChannel 0 configuration	unused
InChannel 0 checks	Plausibility, cut line, short
InChannel 1 configuration	unused
InChannel 1 checks	Plausibility, cut line, short
InChannel 2 configuration	unused
InChannel 2 checks	Plausibility, cut line, short
InChannel 3 configuration	unused
InChannel 3 checks	Plausibility, cut line, short
InChannel 4 configuration	unused
InChannel 4 checks	Plausibility, cut line, short
InChannel 5 configuration	unused
InChannel 5 checks	Plausibility, cut line, short
InChannel 6 configuration	unused
InChannel 6 checks	Plausibility, cut line, short
InChannel 7 configuration	unused
InChannel 7 checks	Plausibility, cut line, short
OutChannel 0 configuration	unused

The setting of the parameters is completed now. All other settings can be left as is. Exit the configuration of the "PDP22-FBP (DPV1 modular)" and the modules by clicking on "OK".

After the configuration of the PROFIBUS slaves has been finished, the PROFIBUS station addresses can be set. This is carried out using the configuration window of the PROFIBUS master. Call the master configuration once more and change to the folder "Configuration/Station table".


netDevice - Configuration AC500_DPM1[CM572-DPM]<1>(#1)

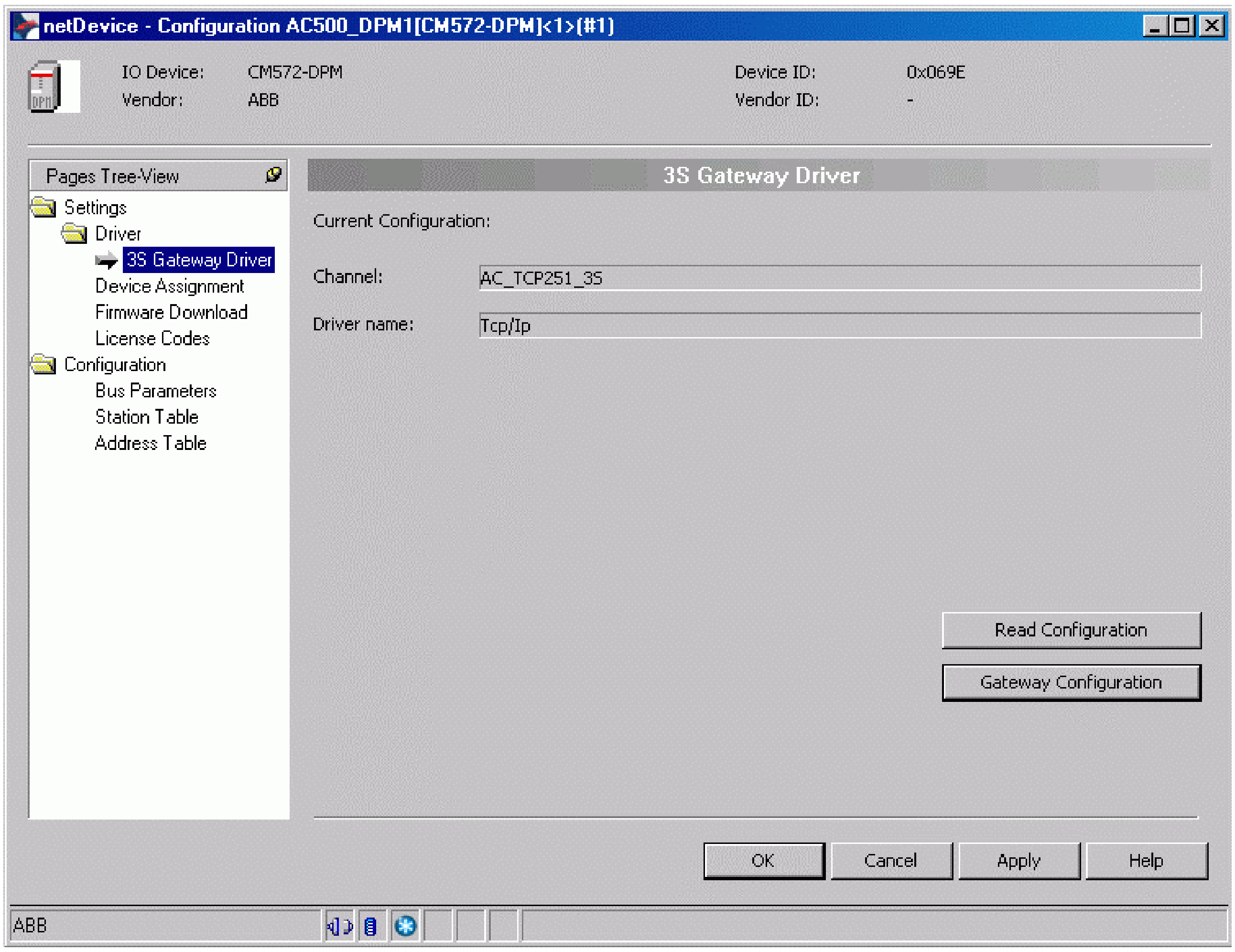
IO Device: CM572-DPM
Vendor: ABB
Device ID: 0x069E
Vendor ID: -

Station Table

Activate	Station ...	Device	Name	Vendor
✓	2	PDP22-FBP (DPV1 modular)	FBP_DPS2	ABB

The station address of the slave can be set now. All other settings can be taken on.

The configuration can be loaded down into the PROFIBUS master now. To do this, the interface must be defined first, via which the data are loaded into the coupler. The corresponding interface can be configured over the configuration window of the PROFIBUS master in the register "Settings/Drivers/3S Gateway Driver".


netDevice - Configuration AC500_DPM1[CM572-DPM]<1>(#1)

IO Device: CM572-DPM
Vendor: ABB
Device ID: 0x069E
Vendor ID: -

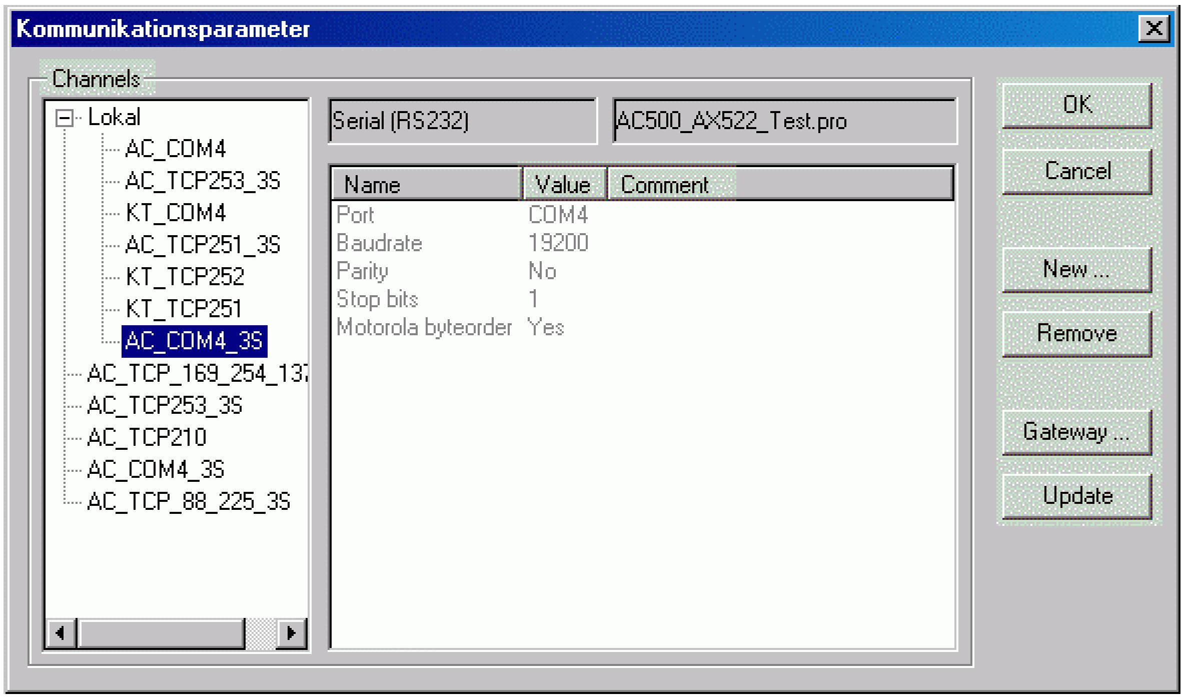
3S Gateway Driver

Current Configuration:

Channel: AC_TCP251_3S

Driver name: Tcp/Ip

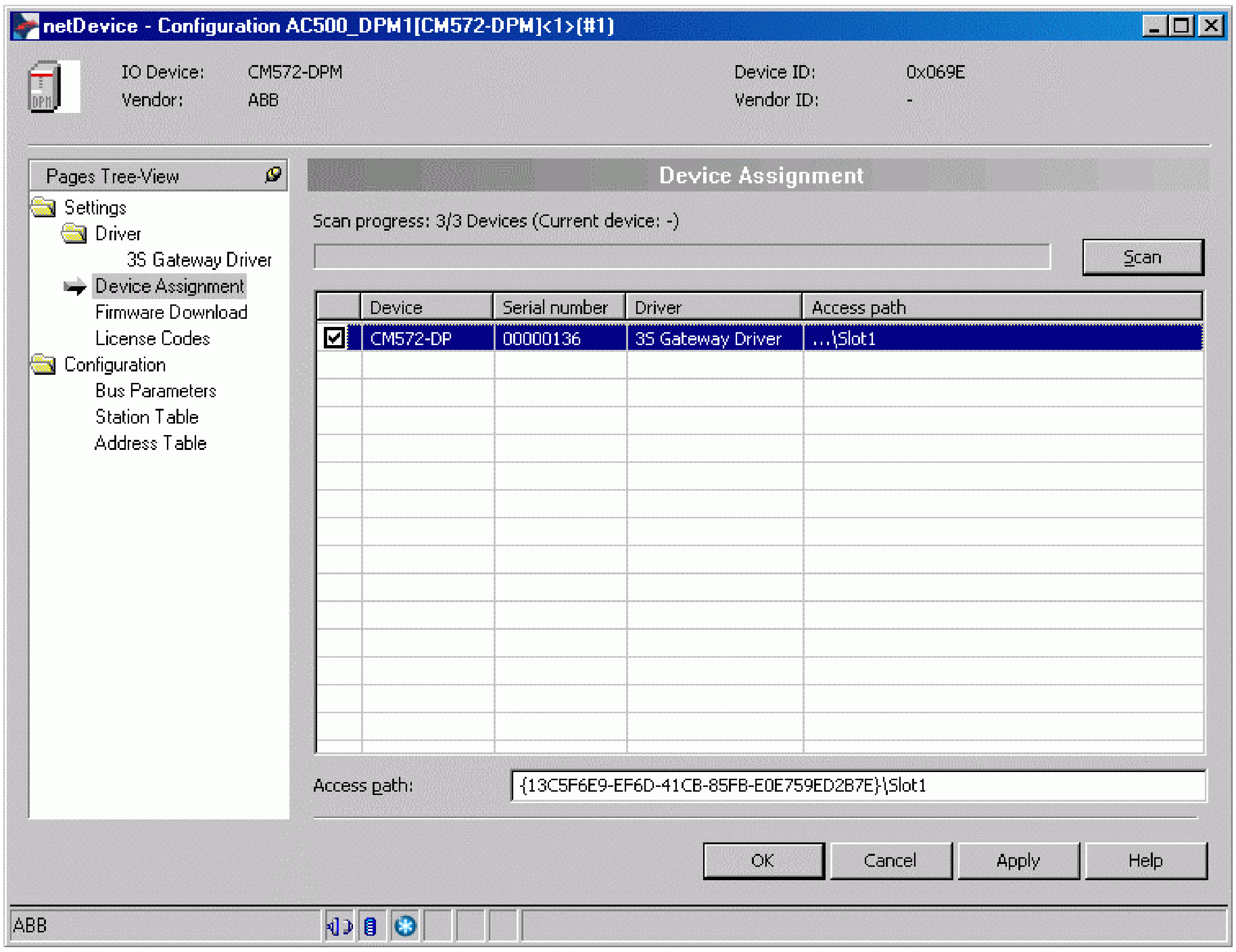
Select "Configure Gateway".


Kommunikationsparameter

Serial (RS232)   AC500_AX522_Test.pro

Name	Value	Comment
Port	COM4	
Baudrate	19200	
Parity	No	
Stop bits	1	
Motorola byteorder	Yes	

The configuration tool SYCON.net is looking now for PROFIBUS couplers which are connected to the indicated interface. A list of the detected PROFIBUS couplers can be looked through over the configuration window of the PROFIBUS master register "Equipment Allocation".


netDevice - Configuration AC500_DPM1[CM572-DPM]<1>(#1)

IO Device: CM572-DPM
Vendor: ABB
Device ID: 0x069E
Vendor ID: -

Device Assignment

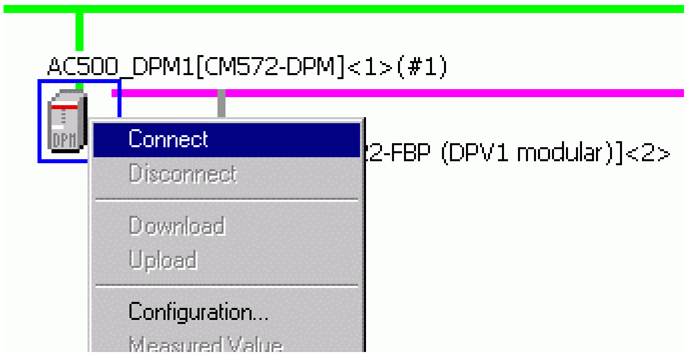
Scan progress: 3/3 Devices (Current device: -)

	Device	Serial number	Driver	Access path
☑	CM572-DP	00000136	3S Gateway Driver	...\Slot1

Access path: {13C5F6E9-EF6D-41CB-85FB-E0E759ED2B7E}\Slot1

Select the representing coupler and confirm with "OK" then.

Move cursor on "CM572", click the right mouse button and "Connect".

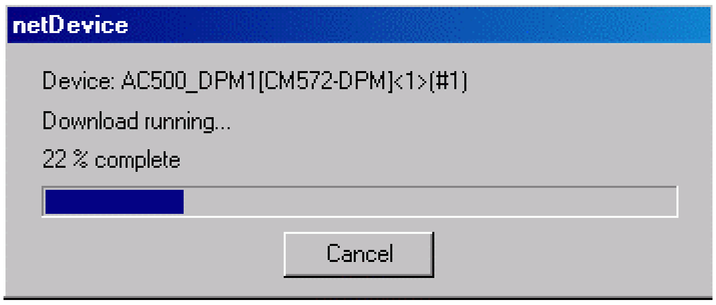


CM572 is shaded green.

Put the control system to Stop.

Move cursor on "CM572", click the right mouse button and "Download".

Confirm query with "Yes".


netDevice

Device: AC500_DPM1[CM572-DPM]<1>(#1)

Download running...

22 % complete

After the successful download the "PWR LED" is on and the "RUN LED" flashes at the CM572.

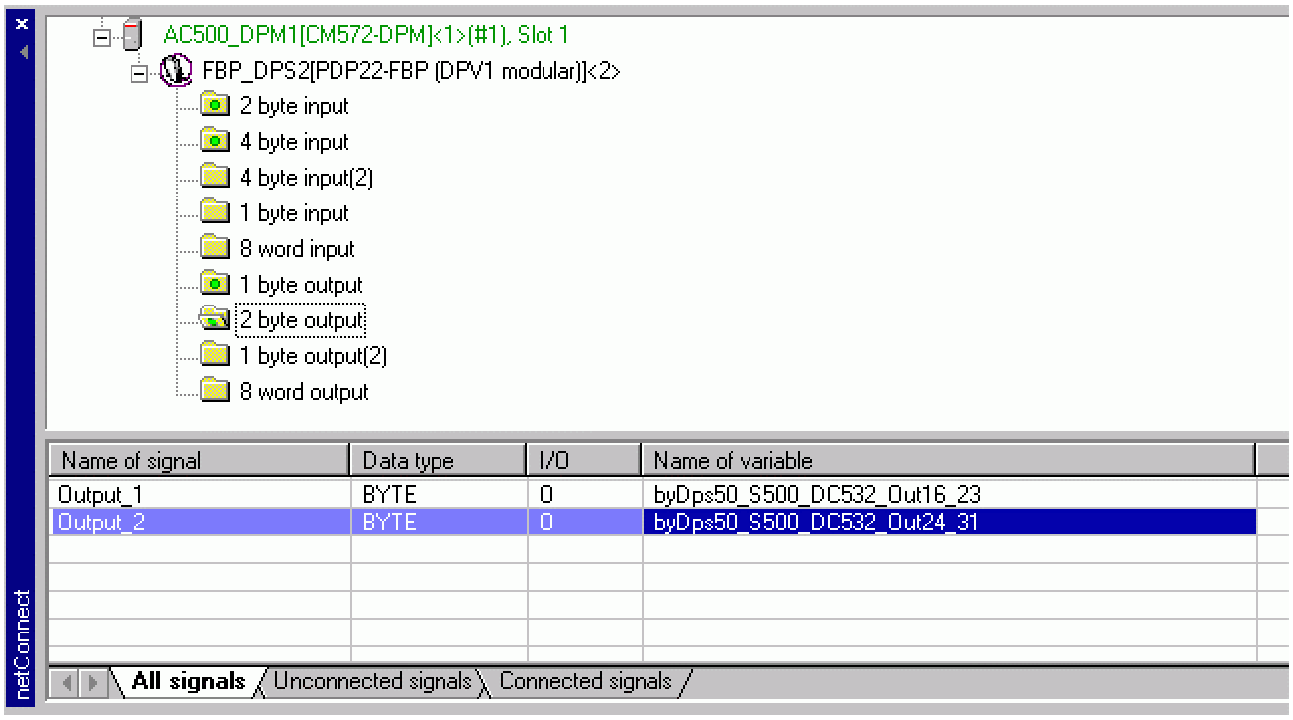
Move cursor on "CM572", click right mouse button and "Disconnect".

The configuration of the coupler is now completed.

The variable names can be listed in the field "Variable Name". A double-click leads into the corresponding entry field.

In order to use the PROFIBUS data in the PLC program, the physical addresses should be assigned corresponding variable names with SYCON.net. These variables are available in the Control Builder then and can be used directly.

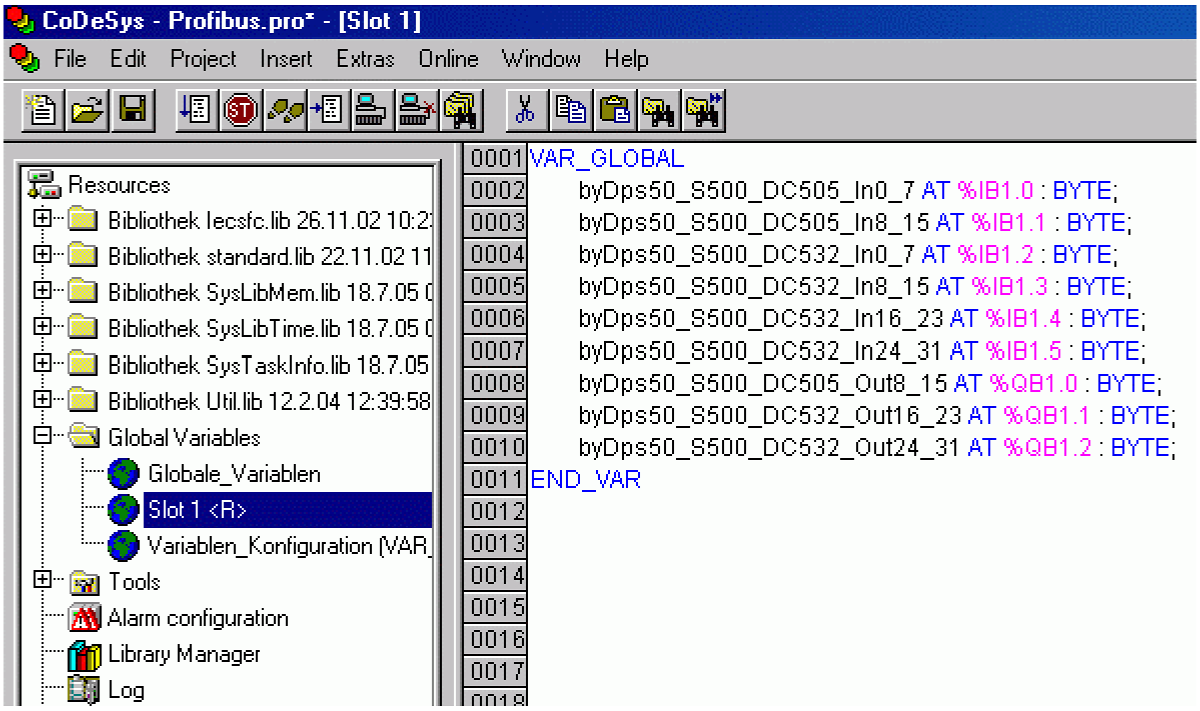
The netConnect window of the SYCON.net is used here.



Name of signal	Data type	I/O	Name of variable
Output_1	BYTE	0	byDps50_S500_DC532_Out16_23
Output_2	BYTE	0	byDps50_S500_DC532_Out24_31

You can enter the variable names in the field "Variable Name". Double click to open the corresponding input field.

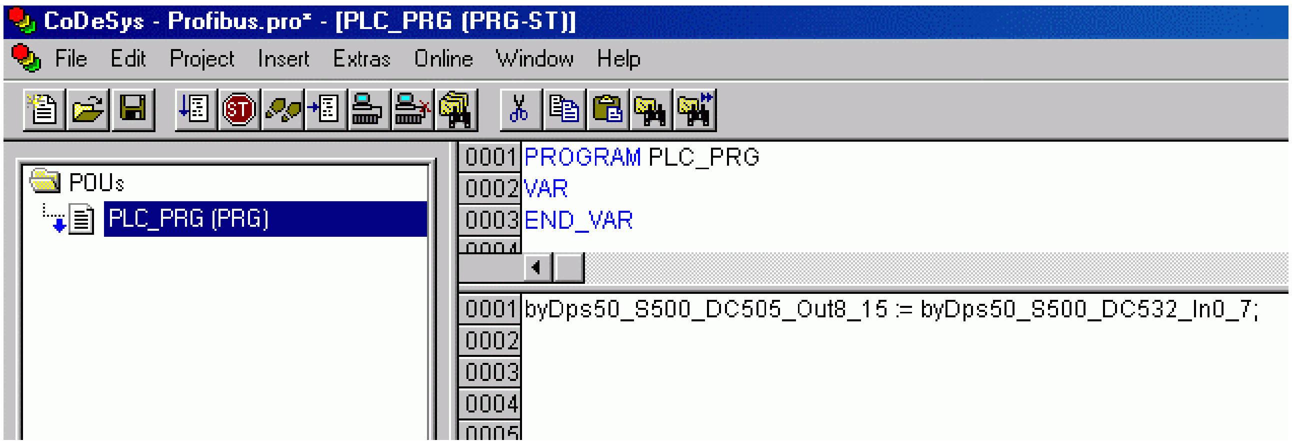
All variables which were declared here are written down on the directory of the "Global Variables" automatically at the focus change from SYCON.net to Control Builder.


CoDeSys - Profibus.pro* - [Slot 1]
File Edit Project Insert Extras Online Window Help
Resources
Bibliothek Iecsfc.lib 26.11.02 10:2
Bibliothek standard.lib 22.11.02 11
Bibliothek SysLibMem.lib 18.7.05 (
Bibliothek SysLibTime.lib 18.7.05 (
Bibliothek SysTaskInfo.lib 18.7.05
Bibliothek Util.lib 12.2.04 12:39:58
Global Variables
Globale_Variablen
Slot 1 <R>
Variablen_Konfiguration (VAR_
Tools
Alarm configuration
Library Manager
Log

```
VAR_GLOBAL
    byDps50_S500_DC505_In0_7 AT %IB1.0 : BYTE;
    byDps50_S500_DC505_In8_15 AT %IB1.1 : BYTE;
    byDps50_S500_DC532_In0_7 AT %IB1.2 : BYTE;
    byDps50_S500_DC532_In8_15 AT %IB1.3 : BYTE;
    byDps50_S500_DC532_In16_23 AT %IB1.4 : BYTE;
    byDps50_S500_DC532_In24_31 AT %IB1.5 : BYTE;
    byDps50_S500_DC505_Out8_15 AT %QB1.0 : BYTE;
    byDps50_S500_DC532_Out16_23 AT %QB1.1 : BYTE;
    byDps50_S500_DC532_Out24_31 AT %QB1.2 : BYTE;
END_VAR
```

The declaration of the variables is completed now. The coupler variables can be used now in the user program.

For the test of the PROFIBUS configuration, the inputs "In 0...7" of the module DC532 are copied to the outputs "Out 8...15" of the module DC505 in the user program.


CoDeSys - Profibus.pro* - [PLC_PRG (PRG-ST)]
File Edit Project Insert Extras Online Window Help
POUs
PLC_PRG (PRG)

```
PROGRAM PLC_PRG
VAR
END_VAR
```

```
byDps50_S500_DC505_Out8_15 := byDps50_S500_DC532_In0_7;
```

The user program can now be transmitted and started in the control system.

The exchange of the PROFIBUS data is indicated by a permanent "ON" of the "RUN LED", shown at the CM572.

1.5.2 Running operation

The PROFIBUS-DP protocol is automatically handled by the coupler and the operating system of the controller. The coupler is only active on the bus if it was correctly initialized before and if the user program is running. No connection elements are required for the cyclic exchange of process data via PROFIBUS-DP. Special PROFIBUS-DP functions can be realized using the function blocks of the corresponding PROFIBUS library.

Communication via PROFIBUS is established by the coupler when starting the user program and starts with the initialization of the configured slaves. After its successful initialization, the slave is added to the cyclic process data exchange. The "RDY" LED lights up steadily after at least one slave was successfully taken into operation. If the user program is stopped, the coupler shuts down the PROFIBUS system in a controlled manner.

The DP master operation mode is completely integrated to the operating system of the controller. The transmit or receive data of the slaves can be directly accessed in the corresponding operand areas. Access can be performed either via operands or symbolically. No function blocks are required.

The function block library contains various blocks which can be used e.g. to poll status information of the coupler or to execute specific acyclic PROFIBUS-DP functions. If necessary, these blocks can be inserted additionally.

1.5.3 Error diagnosis

PROFIBUS-DP communication errors are generally indicated by the red "ERR" LED of the coupler. Malfunctions of the PROFIBUS driver or the coupler itself are additionally indicated via the E error flags and the corresponding LEDs of the CPU. Furthermore, the PROFIBUS library provides different function blocks that allow a detailed error diagnosis. Amongst other things, the following information can be polled:

- the condition of the coupler itself,
- a detailed PROFIBUS diagnosis of an individual slave or
- a system diagnosis overview.

1.5.4 Function blocks

Control Builder		**Libraries**	**Remark**
V1.0	V1.0 or later	PROFIBUS_Master_S90_V43.LIB Coupler_AC500_V10.LIB	

Table 1-2: Overview of PROFIBUS libraries

Profibus_Master_AC500_V10.LIB

Group	**Function block**	**Function**
General		
	PROFI_INFO	Reading of coupler information
Status / Diagnosis		
	DPM_STAT	Reading the coupler status
	DPM_SLVDIAG	Reading the detailed PROFIBUS diagnosis of a slave
	DPM_SYSDIAG	Reading the system diagnosis
Controller		
	DPM_CTRL	Sending control commands to slaves
Acyclic reading		
	DPM_READ_INPUT	Reading input data of slaves which are not assigned to the master
	DPM_READ_OUTPUT	Reading output data of slaves which are not assigned to the master

Table 1-3: Function blocks contained in the library PROFIBUS_Master_AC500_V10.LIB

Coupler_AC500_V10.LIB

Contains various internal functions which are used by the corresponding field bus coupler libraries.

1.6 PROFIBUS DP diagnosis

1.6.1 Status LEDs

The following figure shows the positions of the five status LEDs.

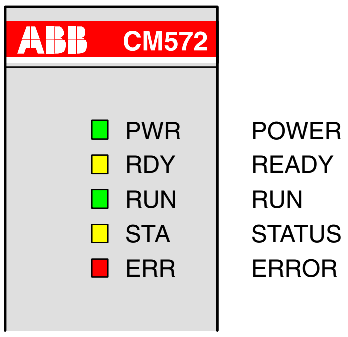


Figure: Positions of the status LEDs

LED	Color	Status	Meaning
PWR	green	on	Supply voltage available
		off	Supply voltage not available
RDY	yellow	on	Coupler ready
		flashes cyclic	Bootstrap loader active
		flashes irregularly	Hardware or system error
		off	Defective hardware
RUN	green	on	Communication is running
		flashes cyclic	Communication stopped
		flashes irregularly	Missing or faulty configuration
		off	No communication
STA	yellow	on	DP slave: data exchange with DP master
			DP master: sending data or token
		off	DP slave: no data exchange
			DP master: no token
ERR	red	on	PROFIBUS error
		off	no error

Table: Meanings of the status LEDs

1.6.2 PROFIBUS-DP error messages

The PROFIBUS error messages are listed in section &sbquo;Error messages of the couplers&lsquo;.

1.6.3 Function blocks

PROFIBUS-DP master

DPM_STAT	Reading the coupler status
DPM_SLVDIAG	Reading the detailed PROFIBUS diagnosis of a slave
DPM_SYSDIAG	Reading the system diagnosis

PROFIBUS-DP slave

DPS_STAT	Reading the coupler status

Online diagnosis

The online diagnosis is described in the documentation for the field bus configuration tool SYCON.net .

1.7 Further information

1.7.1 Standardization

- EN 50170
- DIN 19245 part 1
- DIN 19245 part 3

1.7.2 Important addresses

PROFIBUS Nutzerorganisation e.V. (PNO)
Haid-und-Neu-Straße 7
Germany, D-76131 Karlsruhe
Phone: (+49) 721 9658 590
Telefax: (+49) 721 9658 589
Internet: http://www.profibus.com

1.7.3 Terms, definitions and abbreviations

PROFIBUS DP	**PRO**CESS **FI**ELD**BUS** - **D**ECENTRALIZED **P**ERIPHERY
DPM1	DP master (class 1), normal bus master
DPM2	DP master (class 2), commissioning device
DPS	DP slave
GSD	Modules master data
DPV1	Guideline for functional expansions of PROFIBUS DP
PNO	**P**ROFIBUS **N**utzer **O**rganisation (PROFIBUS user organization)

2 Index

System Description

AC500

Scalable PLC
for Individual Automation

System Technology
of the CANopen Couplers

CANopen

Contents "The CANopen Coupler"

1 The CANopen coupler

1.1 Brief overview

1.1.1 Fundamental properties and fields of application

CANopen is a standardized layer 7 protocol used for decentralized industrial automation systems based on the Controller Area Network (CAN) and the CAN Application Layer (CAL). CANopen is based on a communication profile containing the determination of basic communication mechanisms and their descriptions, such as the mechanisms used for exchanging process data in real-time or for sending alarm telegrams. This common communication profile is the basis for the various CANopen device profiles. The device profiles describe the specific functionality and/or the parameters of a device class. Such device profiles are available for the most important device classes used in industrial automation, such as digital and analog I/O modules, sensors, drives, control units, regulators, programmable controllers or encoders. Further device profiles are projected.

The central element of the CANopen standard is the device functionality description in an object directory (OD). The object directory is divided into one general area containing information about the device (e.g. device identification, manufacturer's name, etc.) as well as communication parameters, and the device-specific area describing the particular functionality of the device. These properties of a CANopen module are documented in the form of a standardized "electronic data sheet" (EDS file).

A CANopen network can consist of a maximum of 128 modules, one NMT master and up to 127 NMT slaves. In contrast to the typical master-slave systems (e.g. PROFIBUS systems), the meanings of the terms master and slave are different for CANopen. In operational mode, all modules are independently able to send messages via the bus. Moreover, the master is able to change the operating mode of the slaves. The CANopen master is normally implemented by a PLC or a PC. The bus addresses of the CANopen slaves can be set in the range between 1 and 127. The device address results in a number of identifiers occupied by this module.

CANopen supports transmission rates of 10 kbit/s, 20 kbit/s, 50 kbit/s, 125 kbit/s, 250 kbit/s, 500 kbit/s, 800 kbit/s and 1 Mbit/s. Each CANopen device has at least to support a transmission rate of 20 kbit/s. Other transmission rates are optional.

1.1.2 Communication mechanisms

CANopen distinguishes two basic mechanisms for data transmission: The fast exchange of short process data via Process Data Objects (PDOs) and the access to entries of the Object Directory using Service Data Objects (SDOs). Service Data Objects are primarily used to transmit parameters during device configuration. The transmission of Process Data Objects is normally performed event oriented, cyclic or on request as broadcast objects.

Service Data Objects

Service Data Objects (SDOs) are used to modify Object Directory entries as well as for status requests. Transmission of SDOs is performed as a confirmed data transfer with two CAN objects in the form of a peer-to-peer connection between two network nodes. The corresponding Object Directory entry is addressed by specifying the index and the sub-index of the entry. It is possible to transmit messages of unlimited length. If necessary, the data are segmented into several CAN messages.

Process Data Objects

For the transmission of process data, the Process Data Object (PDO) mechanism is available. A PDO is transmitted unconfirmed because, in the end, the CAN link layer ensures the error-free transmission. According to the CAN specification, a maximum of 8 data bytes can be transmitted within one PDO. In conjunction with a synchronization message, the transmission as well as the take over of PDOs can be synchronized over the entire network (synchronous PDOs). The assignment of application objects to a PDO can be set using a structural description (PDO mapping) that is stored in the object directory. Thus, an adaptation according to the requirements of the individual applications is possible. The transmission of process data can be performed by various methods.

Event

The PDO transmission is controlled by an internal event, e.g. by a changing level of a digital input or by an expiring device-internal timer.

Request

In this case, another bus subscriber is requesting the process data by sending a remote transmission request (RTR) message.

Synchronous

In case of synchronous transmission, synchronization telegrams are sent by a bus subscriber. These telegrams are received by a PDO producer which in turn transmits the process data.

1.1.3 Network Management

Within a CANopen network, only one NMT master exists (NMT = Network management). All other modules are NMT slaves. The NMT master completely controls all modules and is able to change their states. The following states are distinguished:

Initialization

After switching-on, a node is first in the initialization state. During this phase, the device application and the device communication are initialized. Furthermore, a so-called boot-up message is transmitted by the node to signalize its basic readiness for operation. After this phase is finished, the node automatically changes to the pre-operational state.

Pre-operational

In this state, communication with the node is possible via Service Data Objects (SDOs). The node is not able to perform PDO communication and does not send any emergency messages.

Prepared

In the prepared state, a node is completely disconnected from the network. Neither SDO communication nor PDO communication is possible. A state change of the node can only be initiated by a corresponding network command (e.g. Start Node).

1.1.4 Emergency messages

Emergency messages are used to signalize device errors. An emergency message contains a code that clearly identifies the error (specified in the communication profile DS-301 and in the individual device profiles DS-40x). The following table shows some of the available error codes. Emergency messages are automatically sent by all CANopen modules.

Emergency error code (hex)	Meaning / error cause
00xx	Error on reset or no error
10xx	General error
20xx	Current error
21xx	- Error on device input side
22xx	- Error inside the device
23xx	- Error on device output side
30xx	Voltage error
31xx	- Supply voltage error
32xx	- Error inside the device
33xx	- Error on device output side
40xx	Temperature error
41xx	- Ambient temperature
42xx	- Temperature inside the device
50xx	Hardware error in the device
60xx	Software error in the device
61xx	- Device-internal software
62xx	- Application software
63xx	- Data
70xx	Error in additional modules
80xx	Monitoring
81xx	Communication
90xx	External error
F0xx	Error of additional functions
FFxx	Device-specific errors

Table: Error codes in emergency messages

1.1.5 Node guarding and heartbeat

Testing the functionality of a CAN node is particularly required if the node does not continuously send messages (cyclic PDOs). Two mechanisms can alternatively be used to monitor the CANopen nodes. When the node guarding protocol is used, the NMT master sends messages to the available CANopen slaves which have to respond to these messages within a certain time period. Therefore, the NMT master is able to detect if a node fails. Furthermore, the heartbeat protocol can be used with CANopen. In this case, each node automatically sends a periodic message. This message can be monitored by each other subscriber in the network.

1.1.6 Object directory

The object directory describes the entire functionality of a CANopen device. It is organized as a table. The object directory does not only contain the standardized data types and objects of the CANopen communication profile and the device profiles. If necessary, it also contains manufacturer-specific objects and data types. The entries are addressed by means of a 16 bit index (table row, 65536 entries max.) and an 8 bit sub-index (table column, 256 entries max.). Thus, objects belonging together can be easily grouped. The following table shows the structure of this CANopen object directory:

Index		Object
dec	**hex**	
0	0000	not used
1...31	0001...001F	Static data types
32...63	0020...003F	Complex data types
64...95	0040...005F	Manufacturer-specific data types
96...127	0060...007F	Profile-specific static data types
128...159	0080...009F	Profile-specific complex data types
160...4095	00A0...0FFF	Reserved
4096...8191	1000...1FFF	Communication profile (DS-301)
8192...24575	2000...5FFF	Manufacturer-specific parameters
24576...40959	6000...9FFF	Parameters of the standardized device profiles
40960...65535	A000...FFFF	Reserved

Table: Structure of the object directory

Several data types are defined for the objects themselves. If required, other structures (e.g. ARRAY, STRUCT) can be created from these standard types.

1.1.7 Identifiers

CANopen always uses identifiers with a length of 11 bits (standard frames). The number of available and allowed identifiers given by this is divided into several ranges by the pre-defined connection set. This structure is designed in a way that a maximum of 128 modules (1 NMT master and up to 127 slaves) can exist in a CANopen network. The list of identifiers is composed of some fix identifiers (e.g. network management identifier 0) and various functional groups where each existing node, that supports the corresponding function, is assigned to one unique identifier (e.g. Receive PDO 1 of node 3 = 512 + node number = 515). Using the pre-defined connection set therefore avoids double assignment of identifiers.

Identifier	Function	Calculation
0	Network management (NMT)	-
1...127	not used	-
128	Synchronization (SYNC)	-
129...255	Emergency message	128 + node ID
256	Timestamp message	-
257...384	not used	-
385...511	Transmit PDO 1	384 + node ID
512	not used	-
513...639	Receive PDO 1	512 + node ID
640	not used	-
641...767	Transmit PDO 2	640 + node ID
768	not used	-
769...895	Receive PDO 2	768 + node ID
896	not used	-
897...1023	Transmit PDO 3	896 + node ID
1024	not used	-
1025...1151	Receive PDO 3	1024 + node ID
1152	not used	-
1153...1279	Transmit PDO 4	1152 + node ID
1280	not used	-
1281...1407	Receive PDO 4	1280 + node ID
1408	not used	-
1409...1535	Transmit SDO	1408 + node ID
1536	not used	-
1537...1663	Receive SDO	1536 + node ID
1664...1792	not used	-
1793...1919	NMT error (node guarding, heartbeat, boot-up)	1792 + node ID
1920...2014	not used	-
2015...2031	NMT, LMT, DBT	-

Table: Assignment of identifiers

1.1.8 PDO mapping

As already explained, all 8 data bytes of a CAN message are available for the transmission of process data. As there is no additional protocol information, the data format has to be agreed between the sending (producer) and the receiving party (consumer). This is done by the so-called PDO mapping.

If a fixed mapping is used, the process data are arranged in a pre-defined order within the PDO message. This arrangement is predetermined by the device manufacturer and cannot be changed. If variable mapping is used, the process data can be arranged as desired within the PDO message. For this purpose, the address, consisting of index and sub-index, as well as the size (number of bytes) of an object directory entry are entered into the mapping object.

1.1.9 EDS files

The characteristic properties of a CANopen module are documented in the form of an electronic data sheet (EDS file, electronic data sheet). The file completely and clearly describes the characteristics (objects) of a module type in a standardized and manufacturer independent format. Programs for configuring a CANopen network use the module type descriptions available in the EDS files. This strongly simplifies the configuration of a CANopen system. Usually the EDS files are provided by the device manufacturer.

1.1.10 Features

CANopen:

- Operating mode CANopen-Master
- Process image with a maximum of 57344 I/O points
- Supports min. boot-up, emergency messages and life guarding
- Supported PDO modes: Event-controlled, synchronous, cyclic and remote PDO transmission
- Integrated device profiles: CiA DS-401, CiA DS-402 and CiA DS-406

CAN (additional functionality, not necessary for pure CANopen operation):

- Support of 11 bit identifiers according to CAN 2.0 A and 29 bit identifiers according to CAN 2.0 B
- Transmission and reception of any CAN telegrams via function blocks in the user program

Transmission technique:

- ISO 11898, potential separated
- Transfer rates of 20 kbit/s, 125 kbit/s, 250 kbit/s, 500 kbit/s and 1 Mbit/s
- Bus length up to 1000 m at 20 kbit/s and up to 40 m at 1 Mbit/s
- One bus can have up to 128 subscribers (master + 127 slaves)
- 5-pin COMBICON socket for bus connection

Communication:

- Message-oriented bus access, CSMA/CA
- Predefined master-slave connections
- 8 bytes of non-fragmented user data, for fragmentation any size is possible
- Synchronization of inputs and/or outputs via synchronous PDOs

Protection functions:

- Message transfer with Hamming distance HD = 6
- CAN fault recognition mechanisms via 15 bit CRC, frame check, acknowledge, bit monitoring and bit stuffing
- Incorrect parameter settings are avoided because bus subscribers with faulty parameters are not included in the user data operation.
- Adjustable behavior on subscriber failure. System continues normal operation and the error is indicated at the master or the entire system is stopped.
- Response monitoring of the subscribers (node guarding)

Diagnosis:

- Status indication with 5 LEDs
  - PWR (green): Supply voltage
  - RDY (yellow): Coupler ready for operation
  - RUN (green): Configuration and communication status
  - STA (yellow): Data transfer
  - ERR (red): CANopen error
- Extensive online diagnosis functions in SYCON.net
- Detailed diagnosis in the user program using function blocks

1.2 Technical data

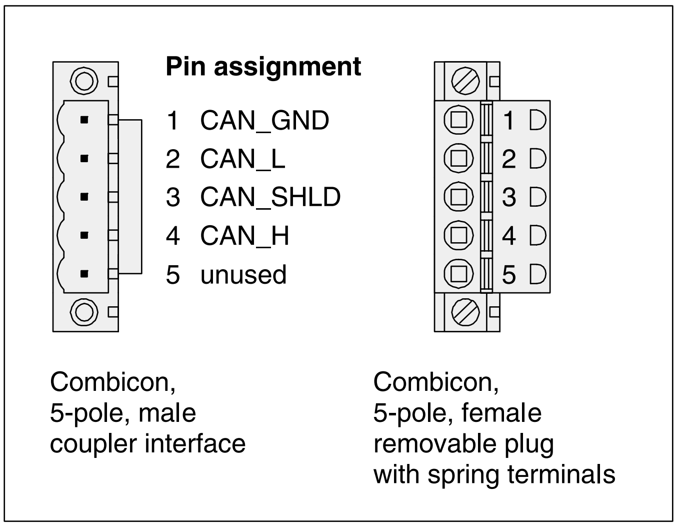
1.2.1 Technical data of the coupler

Coupler CM578-CN	
Field bus	CANopen
Transmission rate	10 kbit/s up to 1 Mbit/s
Protocol	CANopen master
Attachment plug for field bus	Pluggable 5-pin COMBICON connector
Processor	EC1, 160 pins
Clock frequency	48 MHz
Possible CPUs	PM571-xxx, PM581-xxx, PM591-xxx
Possible terminal bases	All
Ambient temperature	0 °C...55 °C
Coupler interface	Dual-port memory, 8 kbytes
Current consumption over the coupler bus	typ. 290 mA
Internal RAM memory (EC1)	256 kbytes
External RAM memory	-
External Flash memory	512 kbytes (firmware)
Status display	PWR, RDY, RUN, STA, ERR
Weight	approx. 150 g

Table: Technical data of the CANopen coupler

1.2.2 Technical data of the interface

Interface socket	5-pin COMBICON
Transmission standard	ISO 11898, potential-free
Transmission protocol	CANopen (CAN), 1 Mbaud max.
Transfer rate (baud rate)	20 kbit/s, 125 kbit/s, 250 kbit/s, 500 kbit/s and 1 Mbit/s
Status indication	5 LEDs (see following figure)
Number of subscribers	127 slaves max.



Pin assignment
1 CAN_GND
2 CAN_L
3 CAN_SHLD
4 CAN_H
5 unused
1
2
3
4
5
Combicon, 5-pole, male coupler interface
Combicon, 5-pole, female removable plug with spring terminals

Figure: Pin assignment of the interface

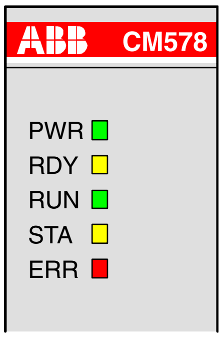


Figure: Position of the LEDs

LED	Color	Status	Meaning
PWR	green	ON	Voltage on
		OFF	Voltage off
RDY	yellow	ON	Coupler ready
		flashes cyclic	Bootstrap loader active
		flashes non-cyclic	Hardware or system error
		OFF	Defective hardware
RUN	green	ON	Communication in progress
		flashes cyclic	Ready for communication
		flashes non-cyclic	Parameterization error
		OFF	No communication or voltage off
STA	yellow	ON	CANopen master: Sends data
		OFF	CANopen master: No data
ERR	red	ON	CANopen error
		OFF	No error

Table: Meaning of the LED states

1.3 Connection and data transfer media

1.3.1 Attachment plug for the bus cable

Assignment:

5-pin COMBICON connector

Pin No.	Signal	Meaning
1	CAN_GND	CAN reference potential
2	CAN_L	Bus line, receive/transmit line, LOW
3	CAN_SHLD	Shield of the bus line
4	CAN_H	Bus line, receive/transmit line, HIGH
5	-	-

Table: Pin assignment of the attachment plug for the bus cable

Supplier:

e.g. COMBICON

Phoenix Contact GmbH & Co.
Flachsmarktstraße 8 - 28
D-32825 Blomberg
Germany
Phone: (+49) (0)52 35 / 3-00
Fax: (+49) (0)52 35 / 3-4 12 00
Internet: http://www.phoenixcontact.com

1.3.2 Bus terminating resistors

The ends of the data lines have to be terminated with a 120 Ω bus termination resistor. The bus termination resistor is usually installed directly at the bus connector.

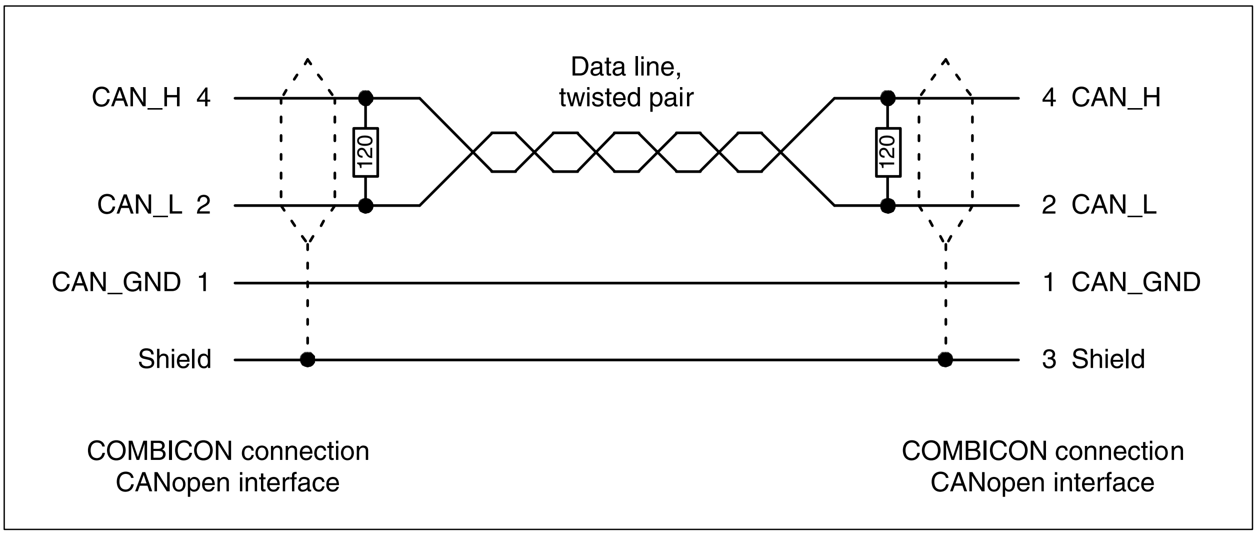


Figure: CANopen interface, bus terminating resistors connected to the line ends

1.3.3 Bus cables

For CANopen, only bus cables with characteristics as recommended in ISO 11898 have to be used. The requirements to the bus cables depend on the length of the bus segment. Regarding this, the following recommendations are given by ISO 11898:

Length of segment [m]	**Bus cable**			**Max. baud rate [kbit/s]**
	Conductor cross section [mm²]	Line resistance [Ω/km]	Wave impedance [Ω]	
0...40	0.25...0.34 / AWG23, AWG22	70	120	1000 at 40 m
40...300	0.34...0.60 / AWG22, AWG20	< 60	120	< 500 at 100 m
300...600	0.50...0.60 / AWG20	< 40	120	< 100 at 500 m
600...1000	0.75...0.80 / AWG18	< 26	120	< 50 at 1000 m

Table: Recommendations for bus cables

Supplier:

e.g. UNITRONIC® BUS CAN

U.I. LAPP GmbH
Schulze-Delitzsch-Straße 25
D-70565 Stuttgart
Germany
Phone: (+49) (0)711 7838 01
Fax: (+49) (0)711 7838 264
Internet: http://www.lappkabel.de

1.4 Possibilities for networking

The CANopen coupler is connected to the bus using the 5-pin COMBICON socket. For EMC suppression and protection against dangerous contact voltages, the shield of the bus line has to be connected to protective earth outside the housing. The line ends of the bus cable have to be terminated using bus terminating resistors.

Within a CANopen network, the controller with the CANopen coupler is the NMT master. No other NMT master is allowed in this network. The NMT master completely controls all modules and their operational states. Up to 127 NMT slaves can be connected to an NMT master.

The CANopen master is able to:

- Change operational states of the slaves
- Parameterize the slaves (e.g. communication connections, time supervision, bus traffic)
- Configure slaves (e.g. type, number and channel operating mode)
- Read input data of the slaves
- Write output data of the slaves
- Read diagnostic data of the slaves
- Monitor the availability of the slaves
- Transmit control commands to synchronize the inputs or outputs of the slaves
- Read and write slave objects even during running operation

The CANopen coupler is as well able to:

- Transmit and receive CAN telegrams according to CAN 2.0 A (11 bit identifier) and CAN 2.0 B (29 bit identifier). (This additional functionality is not required for pure CANopen operation.)

1.5 CANopen implementation

1.5.1 Configuration

The integration of the coupler into the PLC configuration of AC500 is an assumption for the correct function of the CANopen coupler CM578. The configuration of the coupler and the connected CANopen subscribers is done using the tool SYCON.net which is part of the Control Builder programming software.

In the following configuration example, the coupler CM578 is configured as CANopen master device. An I/O device is used as CANopen slave.

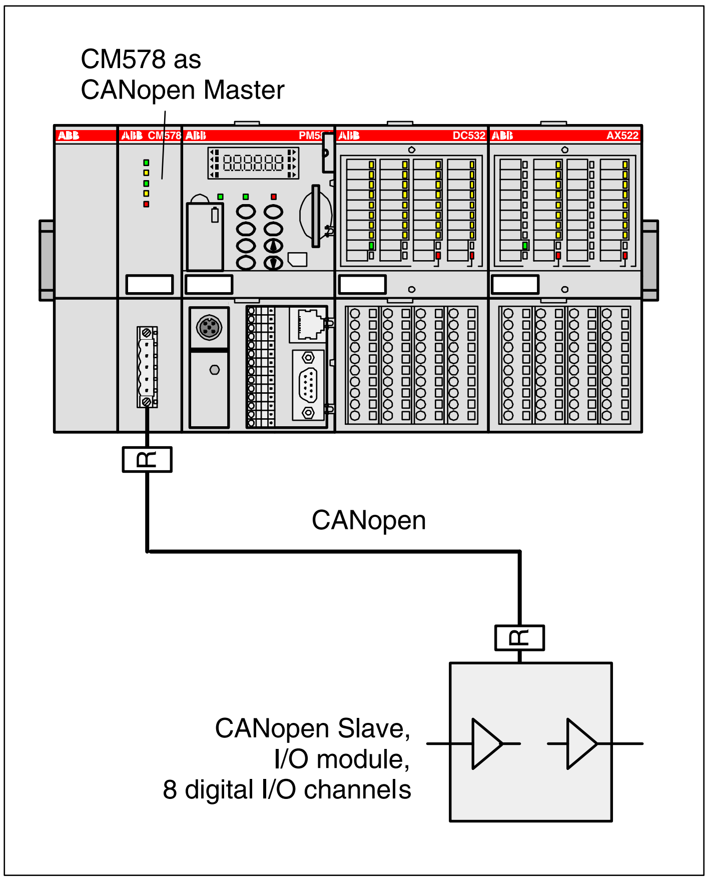
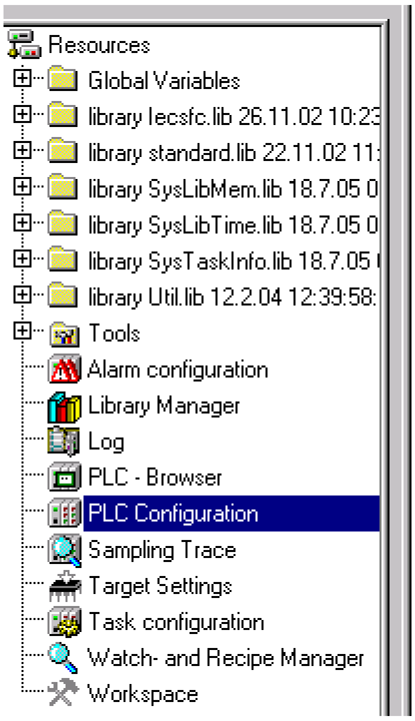


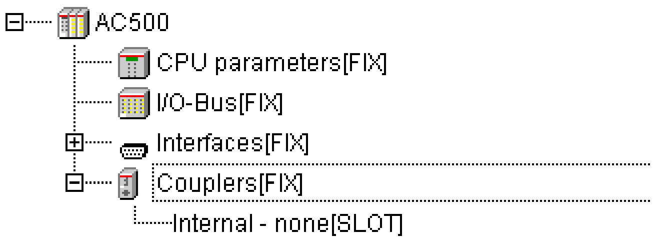
Figure: Example configuration consisting of an AC500 with a CANopen coupler and a CANopen slave

PLC configuration

The configuration of the CANopen coupler starts with the integration of the coupler into the PLC configuration.

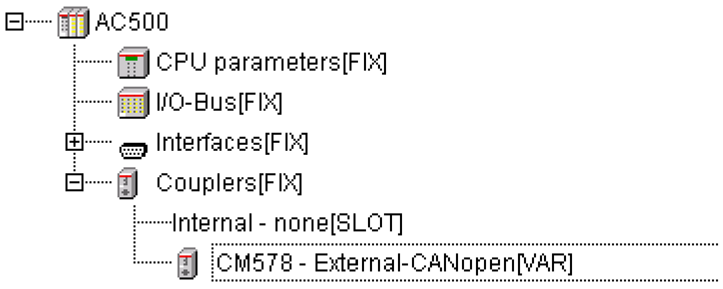


Resources
- Global Variables
- library Iecsfc.lib 26.11.02 10:23
- library standard.lib 22.11.02 11:
- library SysLibMem.lib 18.7.05 0
- library SysLibTime.lib 18.7.05 0
- library SysTaskInfo.lib 18.7.05
- library Util.lib 12.2.04 12:39:58:
- Tools
- Alarm configuration
- Library Manager
- Log
- PLC - Browser
- PLC Configuration
- Sampling Trace
- Target Settings
- Task configuration
- Watch- and Recipe Manager
- Workspace



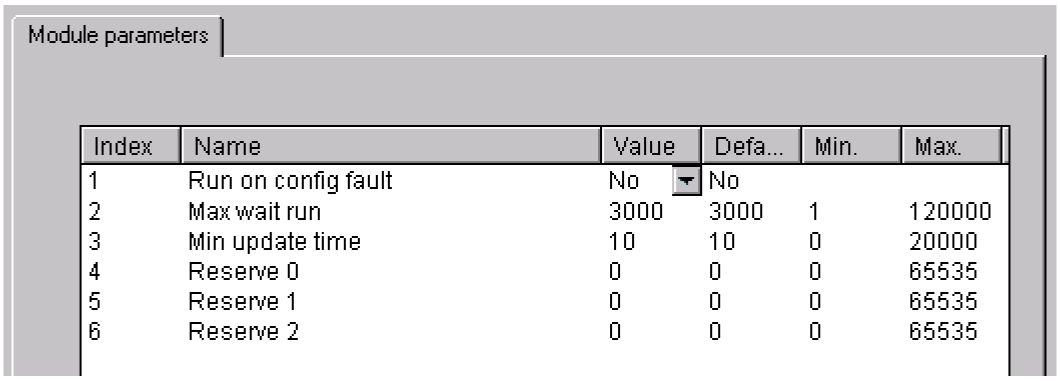
```
AC500
  CPU parameters[FIX]
  I/O-Bus[FIX]
  Interfaces[FIX]
  Couplers[FIX]
    Internal - none[SLOT]
```

To insert the coupler into the configuration, select "Couplers", press the right mouse button and then select "Append Subelement" -> "CM578".



```
AC500
  CPU parameters[FIX]
  I/O-Bus[FIX]
  Interfaces[FIX]
  Couplers[FIX]
    Internal - none[SLOT]
    CM578 - External-CANopen[VAR]
```

Do not change the default values for the coupler parameters.



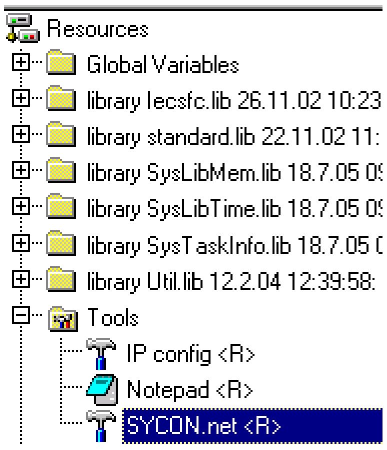
Module parameters

Index	Name	Value	Defa...	Min.	Max.
1	Run on config fault	No	No		
2	Max wait run	3000	3000	1	120000
3	Min update time	10	10	0	20000
4	Reserve 0	0	0	0	65535
5	Reserve 1	0	0	0	65535
6	Reserve 2	0	0	0	65535

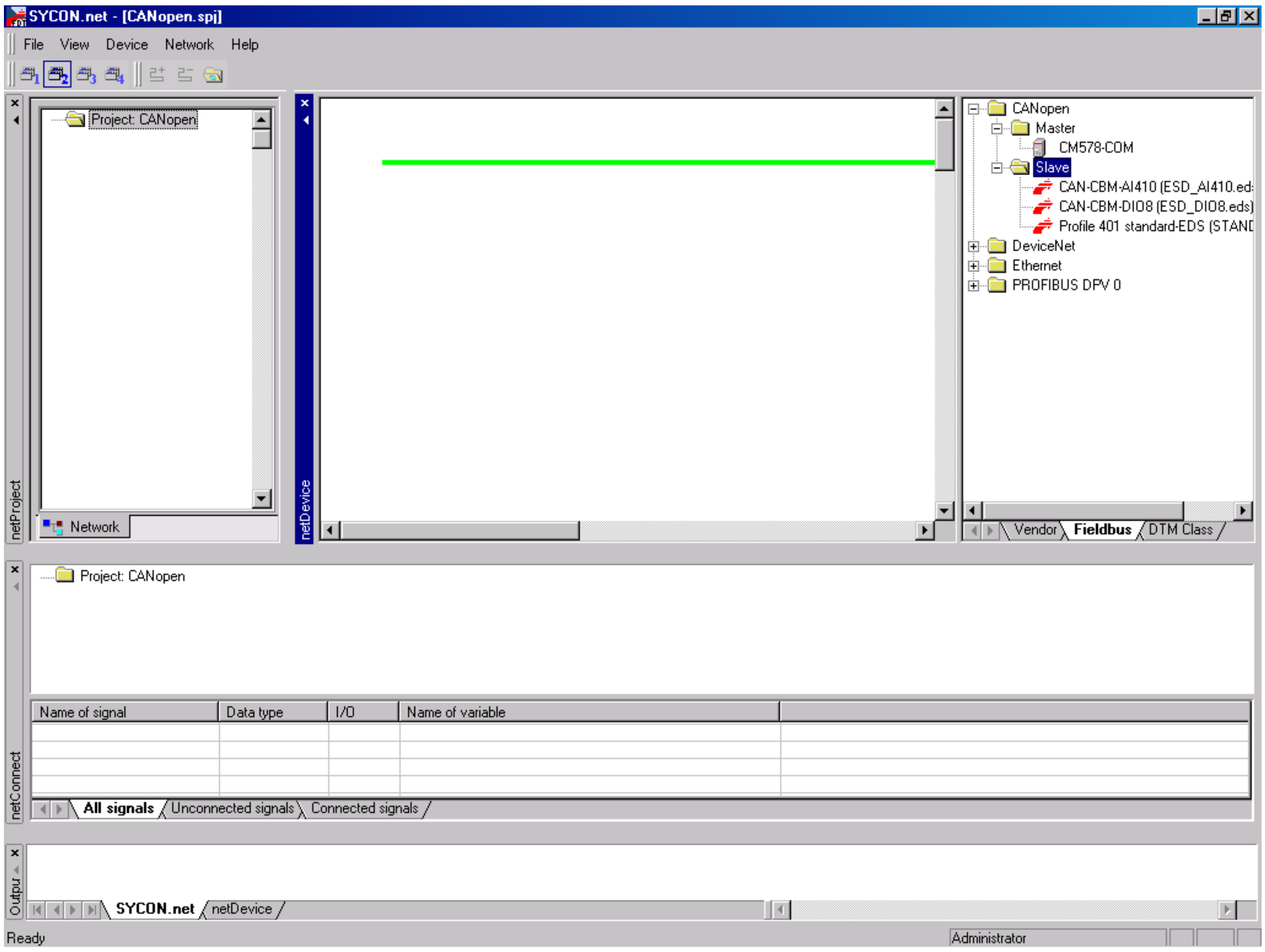
The coupler is now integrated in the PLC configuration.

Configuration using SYCON.net

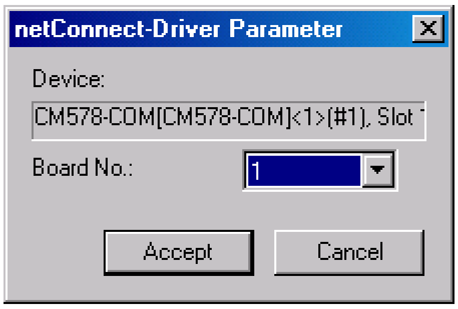
When configuring the CANopen coupler, the configuration data are a definite element of a project. They are specified using the tool SYCON.net (Resources tab -> Tools -> SYCON.net) which is part of the Control Builder. The configuration data are transferred to the coupler with the SYCON.net tool.


Resources
Global Variables
library Iecsfc.lib 26.11.02 10:23
library standard.lib 22.11.02 11:
library SysLibMem.lib 18.7.05 0
library SysLibTime.lib 18.7.05 0
library SysTaskInfo.lib 18.7.05 (
library Util.lib 12.2.04 12:39:58:
Tools
IP config <R>
Notepad <R>
SYCON.net <R>

The following view appears when starting the configuration tool SYCON.net.



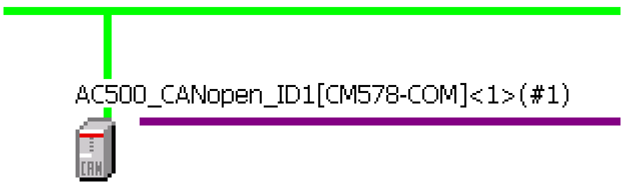
In the top right window, click on the entry "CM578-COM" in the folder "CANopen/Master" and drag it onto the green line displayed in the middle window. Correct insertion positions are displayed by a "+".


netConnect-Driver Parameter

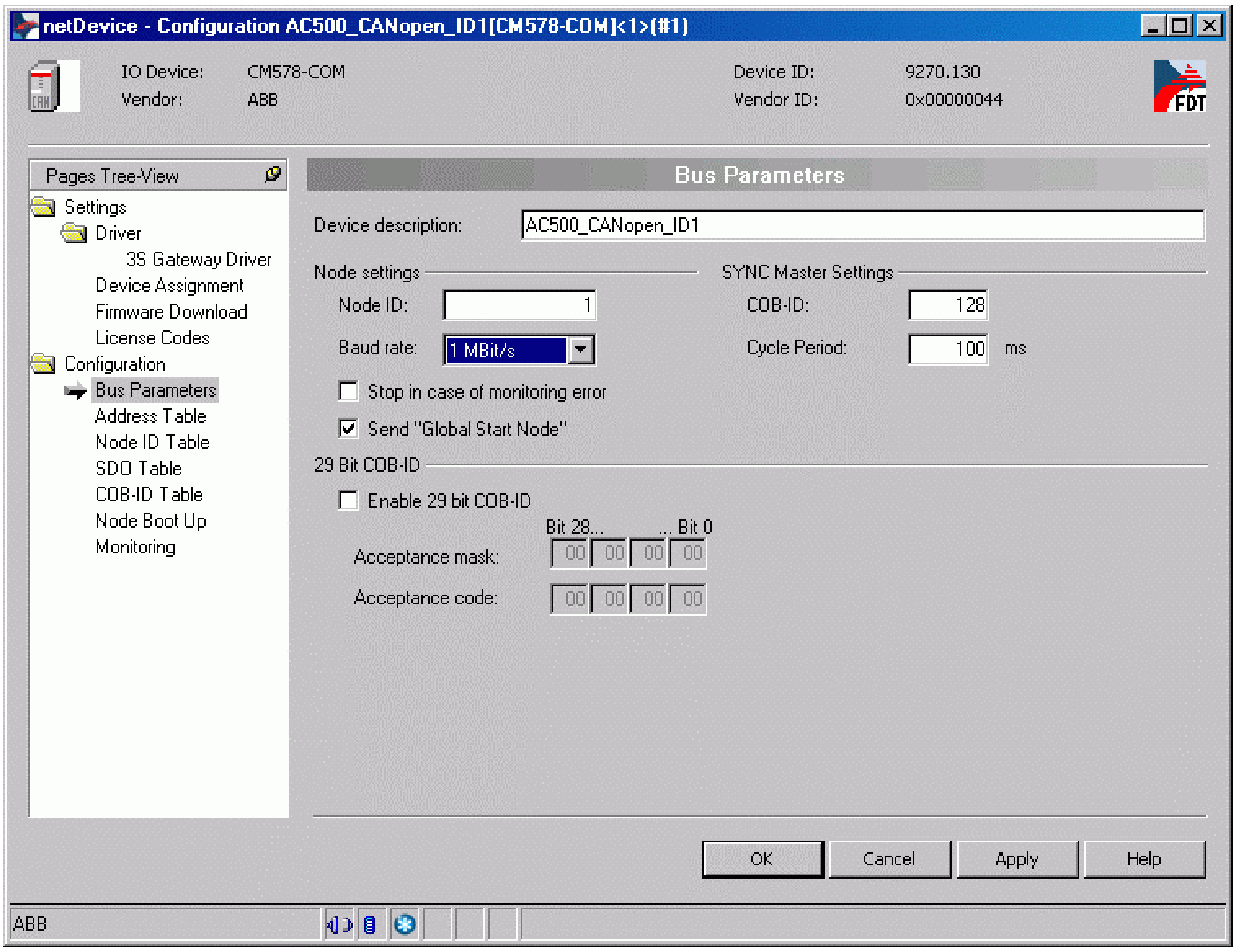
Device: CM578-COM[CM578-COM]<1>(#1), Slot

Board No.: 1

A dialog appears where you have to select the board number according to the coupler slot. The first slot left of the CPU is slot 1 (or board number 1).



To configure the CANopen master, place the cursor on the "CM578" icon and then press the right mouse button and select "Configuration".


netDevice - Configuration AC500_CANopen_ID1[CM578-COM]<1>(#1)

IO Device: CM578-COM
Vendor: ABB
Device ID: 9270.130
Vendor ID: 0x00000044

Bus Parameters

Device description: AC500_CANopen_ID1

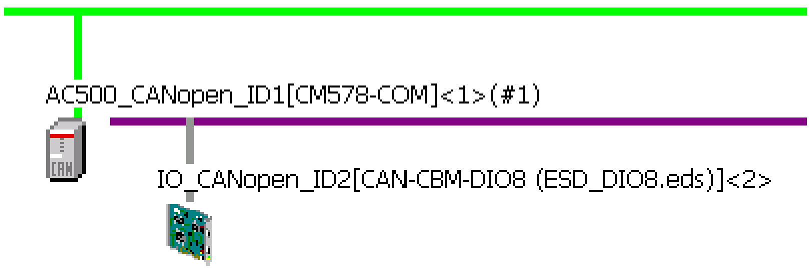
Node settings
Node ID: 1
Baud rate: 1 MBit/s
[ ] Stop in case of monitoring error
[✓] Send "Global Start Node"

SYNC Master Settings
COB-ID: 128
Cycle Period: 100 ms

29 Bit COB-ID
[ ] Enable 29 bit COB-ID
Bit 28... ... Bit 0
Acceptance mask: 00 00 00 00
Acceptance code: 00 00 00 00

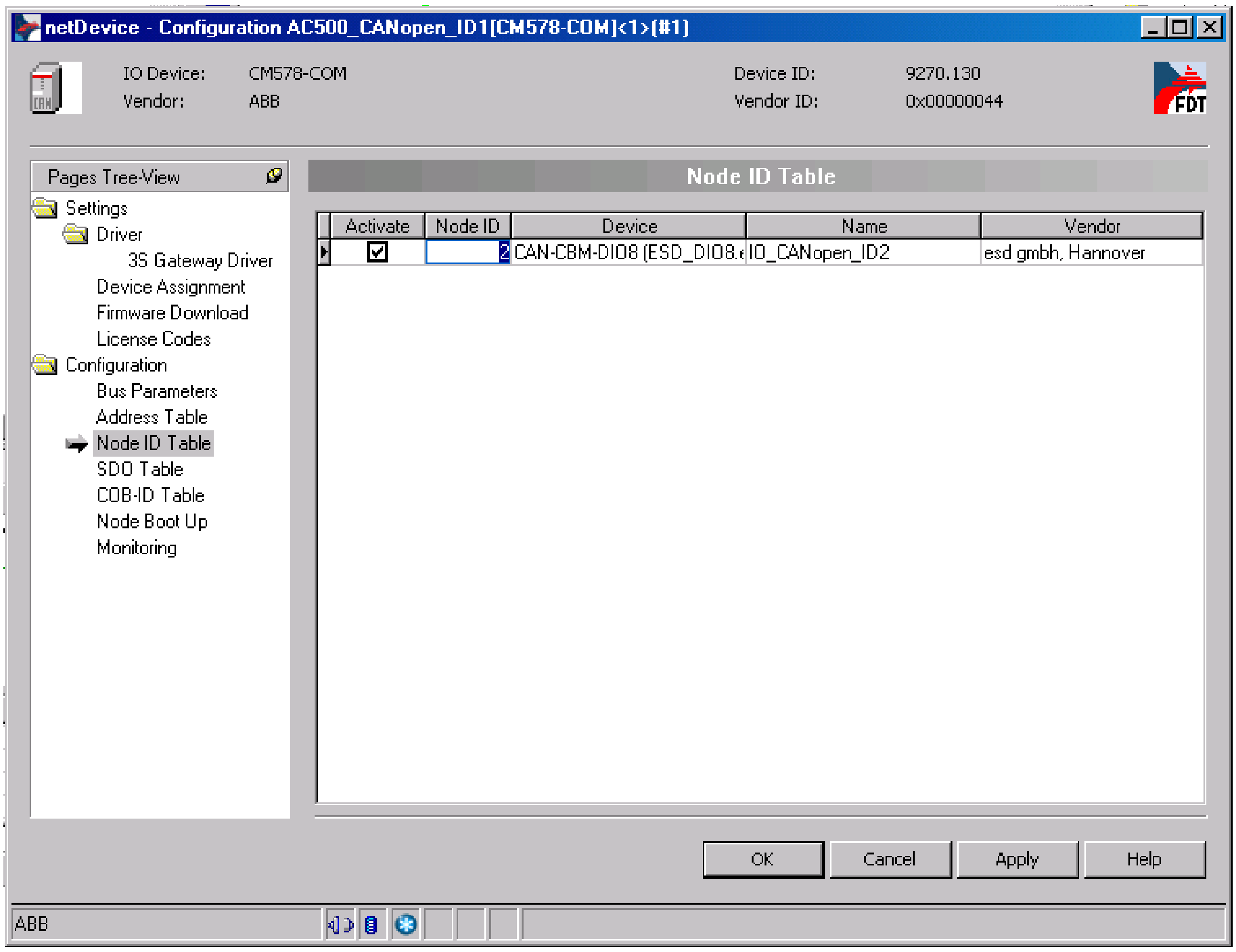
Enter the node ID, select the baud rate of the CANopen coupler and apply the parameters by pressing "OK".

In the top right window, click on the entry "CAN-CBM-DIO8" in the folder "CANopen/Slave" and drag it onto the purple line (CANopen line) displayed in the middle window. Correct insertion positions are displayed by a "+".


AC500_CANopen_ID1[CM578-COM]<1>(#1)
IO_CANopen_ID2[CAN-CBM-DIO8 (ESD_DIO8.eds)]<2>

To configure the CANopen slave, place the cursor on the slave icon and then press the right mouse button and select "Configuration".

In this example, we do not change the configuration of the CANopen slave. Only the node ID of the slave is changed according to the HW setting. This is done in the master configuration dialog. Open the master configuration and select the subitem "Node ID table" in the "Configuration" folder.


netDevice - Configuration AC500_CANopen_ID1[CM578-COM]<1>(#1)

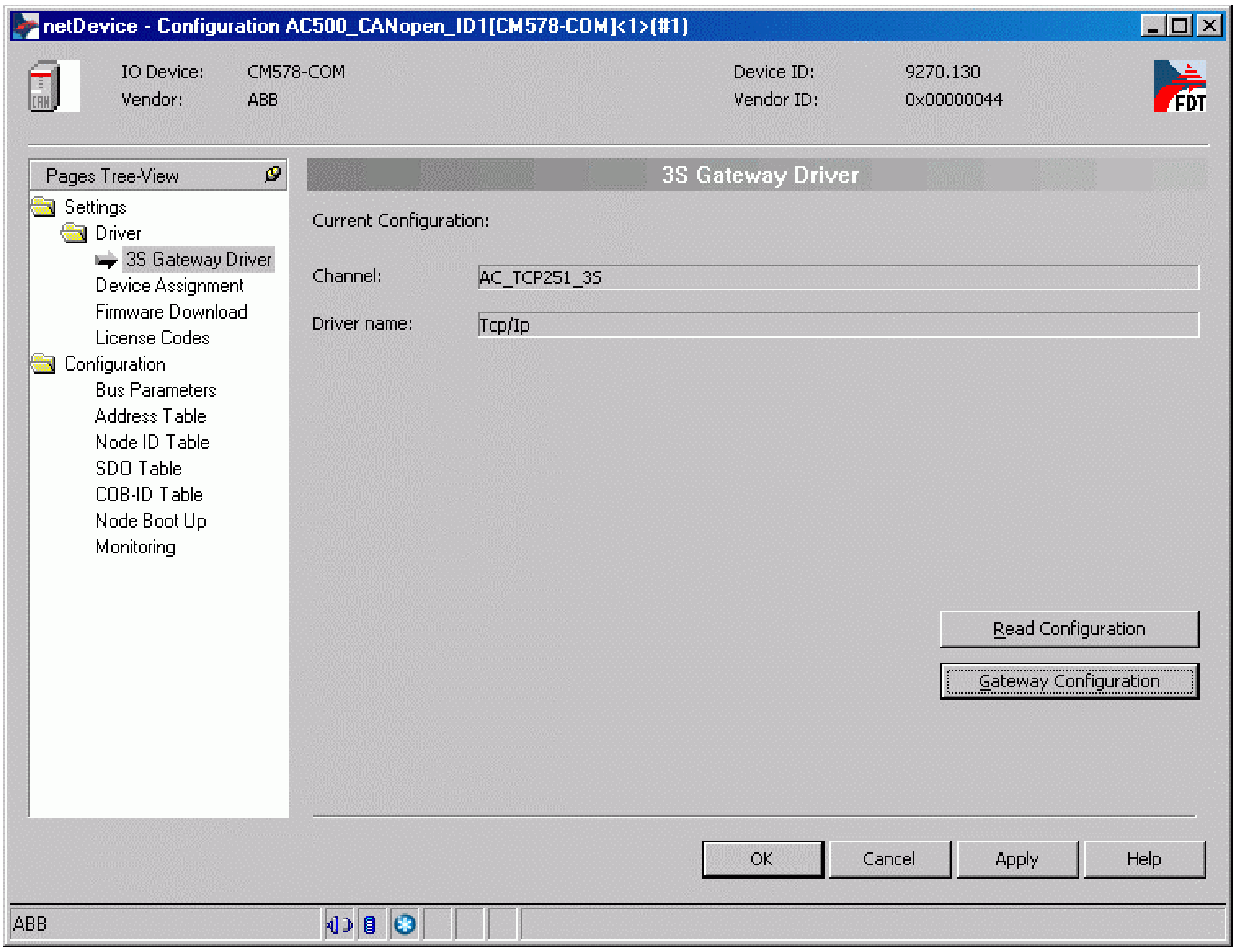
IO Device:	CM578-COM	Device ID:	9270.130
Vendor:	ABB	Vendor ID:	0x00000044

Node ID Table

Activate	Node ID	Device	Name	Vendor
☑	2	CAN-CBM-DIO8 (ESD_DIO8.e	IO_CANopen_ID2	esd gmbh, Hannover

Set the node ID of the slave. Leave all other settings unchanged.

Now the configuration can be downloaded to the CANopen master. To do this, first the interface used to download the data to the coupler has to be specified. The interface to be used is configured in the tab "Settings -> Driver -> 3S Gateway Driver" in the CANopen master configuration.



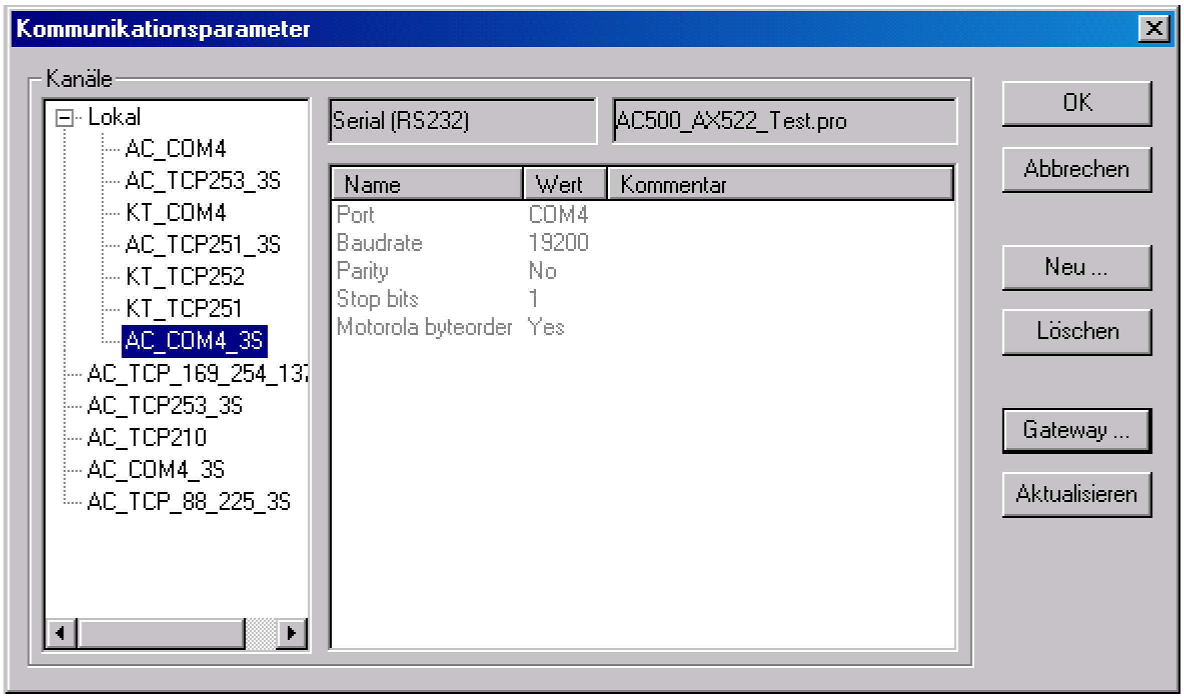
IO Device: CM578-COM
Vendor: ABB
Device ID: 9270.130
Vendor ID: 0x00000044

Current Configuration:

Channel: AC_TCP251_3S

Driver name: Tcp/Ip

Press the button "Gateway Configuration".

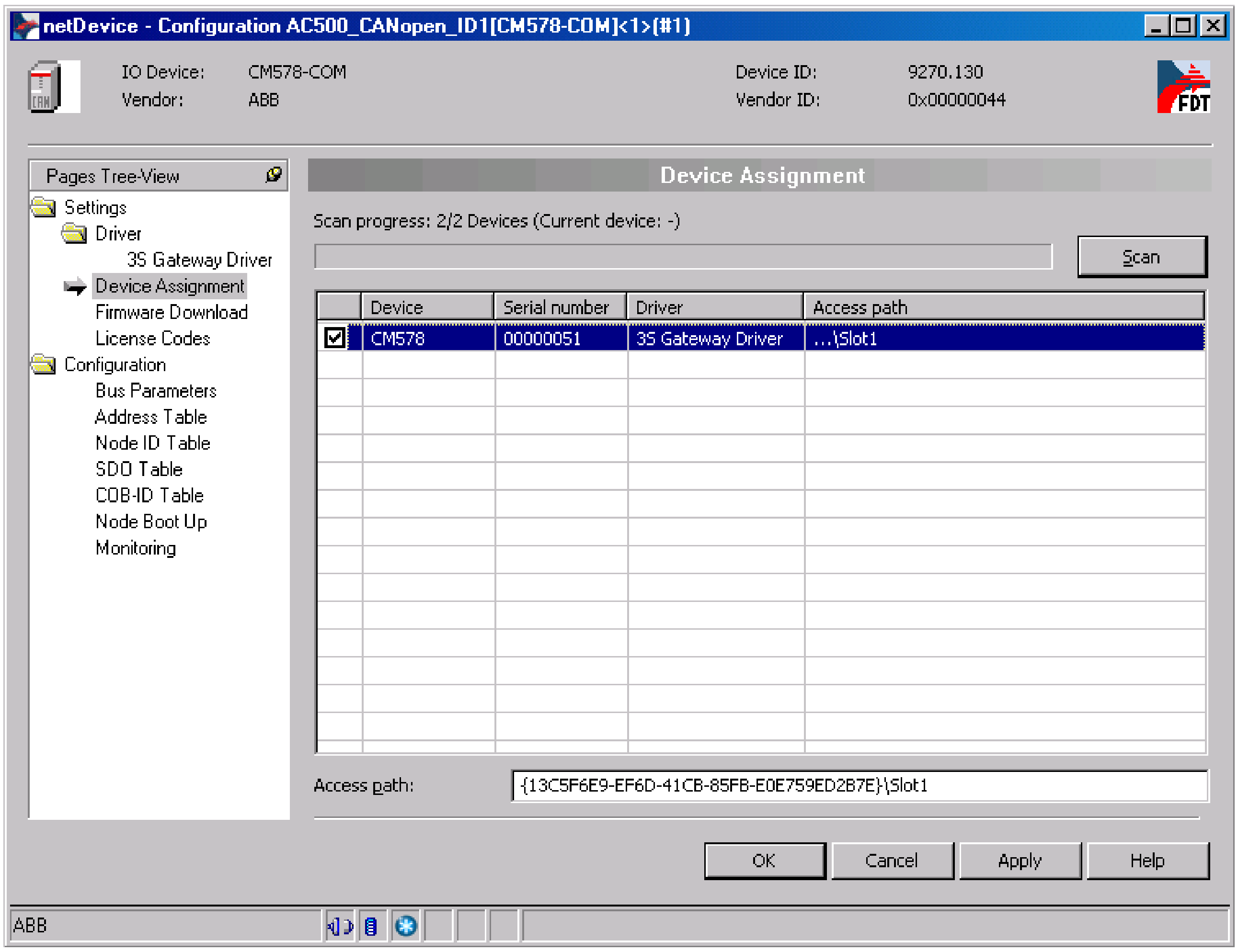


Serial (RS232) | AC500_AX522_Test.pro

Name	Wert	Kommentar
Port	COM4	
Baudrate	19200	
Parity	No	
Stop bits	1	
Motorola byteorder	Yes	

Select the gateway and click on "OK".

Now the configuration tool SYCON.net searches for CANopen couplers that are connected to the selected interface. The tab "Device Assignment" in the CANopen master configuration shows the detected CANopen couplers.


netDevice - Configuration AC500_CANopen_ID1[CM578-COM]<1>(#1)

IO Device:	CM578-COM	Device ID:	9270.130
Vendor:	ABB	Vendor ID:	0x00000044

Device Assignment

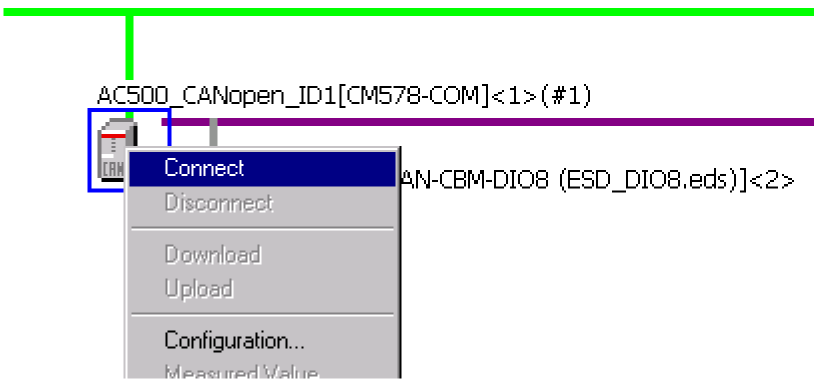
Scan progress: 2/2 Devices (Current device: -)

	Device	Serial number	Driver	Access path
☑	CM578	00000051	3S Gateway Driver	...\Slot1

Access path: {13C5F6E9-EF6D-41CB-85FB-E0E759ED2B7E}\Slot1

Select the desired coupler by marking the relevant checkbox and confirm your selection with "OK".

Position the cursor on the "CM578" icon, press the right mouse button and select "Connect".

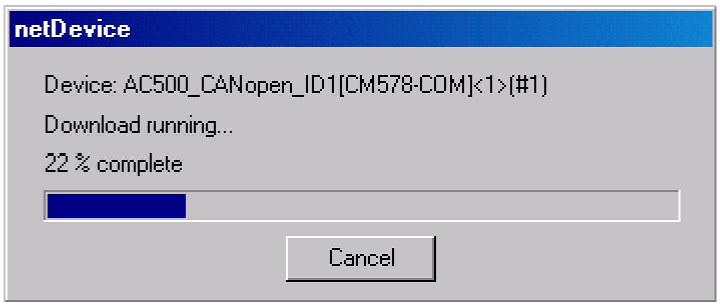

AC500_CANopen_ID1[CM578-COM]<1>(#1)
AN-CBM-DIO8 (ESD_DIO8.eds)]<2>
Connect
Disconnect
Download
Upload
Configuration...
Measured Value

The CM578 icon is then highlighted by green background.

Set the controller to stop state.

Position the cursor on the "CM578" icon, press the right mouse button and select "Download".

Confirm the appearing dialog with "Yes".


netDevice

Device: AC500_CANopen_ID1[CM578-COM]<1>(#1)

Download running...

22 % complete

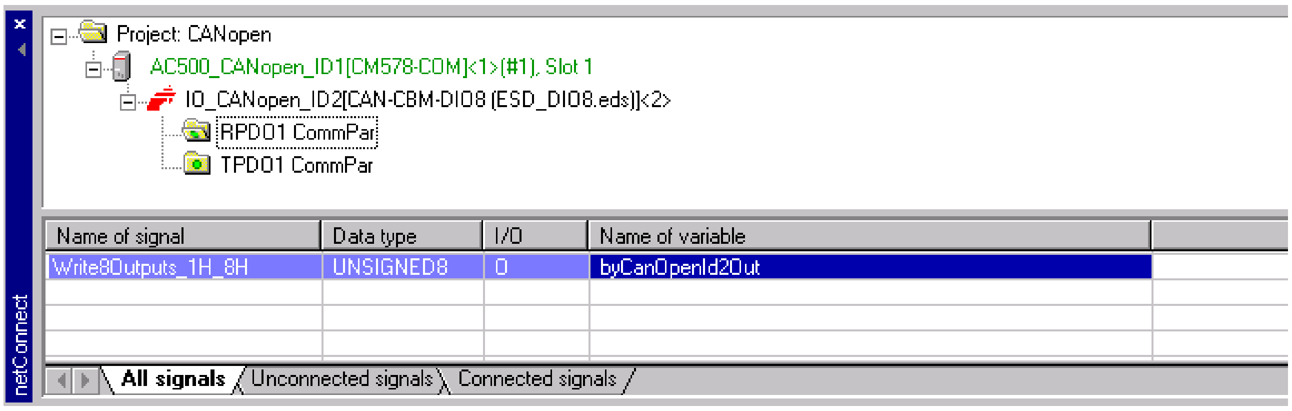
After successful completion of the download process, the "PWR" LED at the CM578 is on and the "RUN" LED flashes.

Position the cursor on the "CM578" icon, press the right mouse button and select "Disconnect".

The coupler configuration is now completed.

To be able to use the CANopen data in the PLC program, you should assign corresponding variable names to the physical addresses using SYCON.net. These variables will then be available in the Control Builder and can be used directly there.

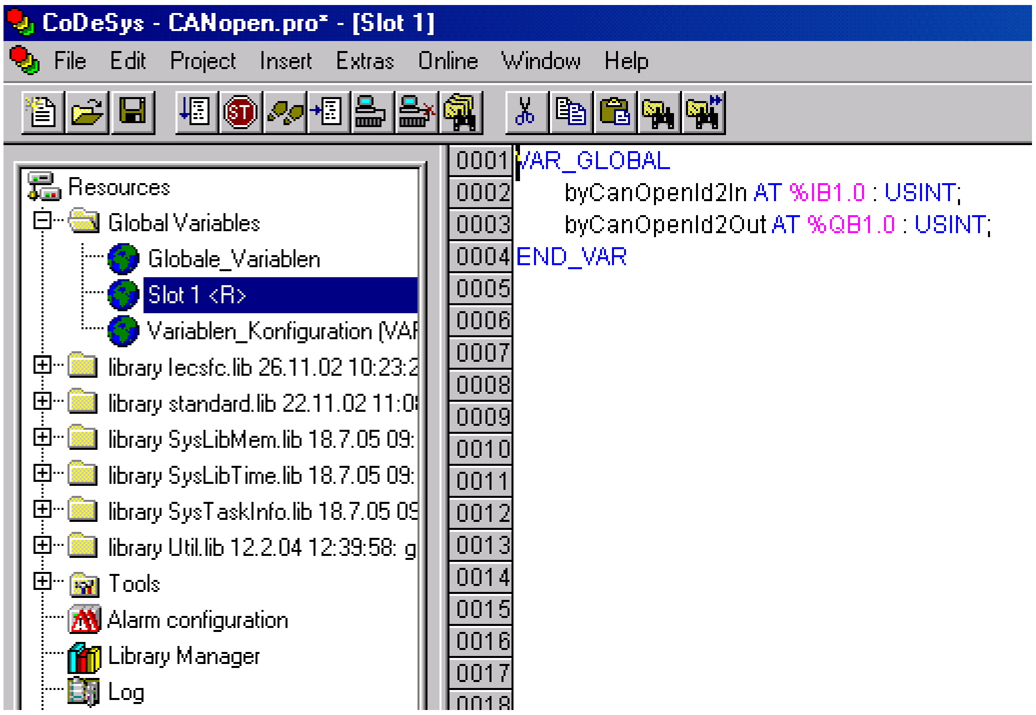
The assignment of the variable names is done in the netConnect window in SYCON.net.



Name of signal	Data type	I/O	Name of variable
Write8Outputs_1H_8H	UNSIGNED8	O	byCanOpenId2Out

The variable name has to be entered in the field "Variable Name". Double click to open the corresponding input field.

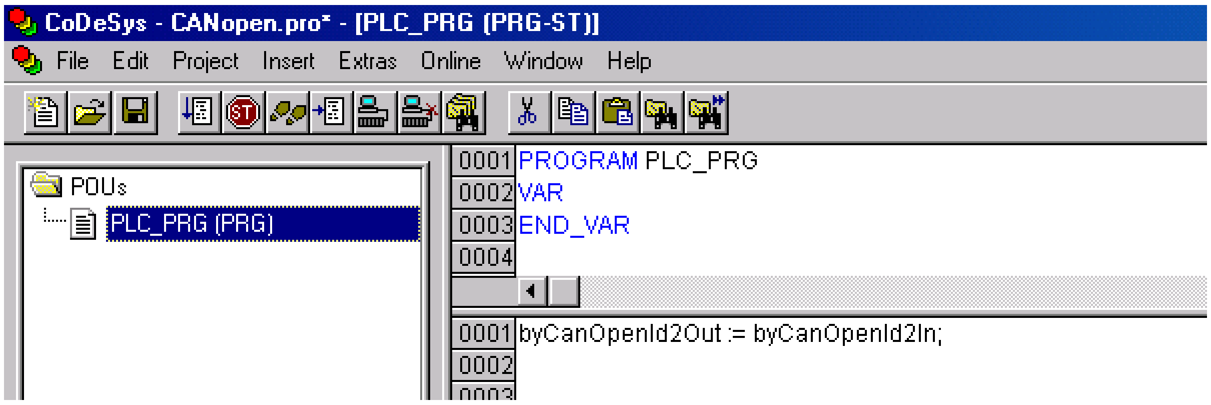
All variables declared here are automatically added to the "Global variables" folder when switching from SYCON.net to the Control Builder.


CoDeSys - CANopen.pro* - [Slot 1]
File
Edit
Project
Insert
Extras
Online
Window
Help
Resources
Global Variables
Globale_Variablen
Slot 1 <R>
Variablen_Konfiguration (VAR
library Iecsfc.lib 26.11.02 10:23:2
library standard.lib 22.11.02 11:0
library SysLibMem.lib 18.7.05 09:
library SysLibTime.lib 18.7.05 09:
library SysTaskInfo.lib 18.7.05 09
library Util.lib 12.2.04 12:39:58: g
Tools
Alarm configuration
Library Manager
Log

```
VAR_GLOBAL
	byCanOpenId2In AT %IB1.0 : USINT;
	byCanOpenId2Out AT %QB1.0 : USINT;
END_VAR
```

The declaration of the variables is now completed. The coupler variables can now be used in the user program.

The CANopen configuration is now tested by copying the CANopen slave ID2 inputs to the CANopen slave ID2 outputs in the user program.


CoDeSys - CANopen.pro* - [PLC_PRG (PRG-ST)]
File Edit Project Insert Extras Online Window Help
POUs
PLC_PRG (PRG)

```
PROGRAM PLC_PRG
VAR
END_VAR
```

```
byCanOpenId2Out := byCanOpenId2In;
```

Now the user program can be downloaded to the controller and started.

During exchange of CANopen data, the "RUN" LED at the CM578 continuously lights up.

1.5.2 Running operation

The CANopen protocol is automatically processed by the coupler and the operating system of the controller. The coupler is only active on the bus if it has been initialized correctly before and if the user program is running. No function blocks are necessary for exchanging process data via CANopen. Special CAN functions can be realized using the function blocks of the CANopen library.

The coupler starts communication via CANopen after the user program is started and then attempts to initialize the configured slaves. After a successful initialization, the slave exchanges the process data. The exchange of I/O data with the slaves is done automatically.

If the user program is stopped, the coupler shuts down the CANopen communication in a controlled manner.

1.5.3 Error diagnosis

CANopen communication errors are indicated by the coupler LEDs. Malfunctions of the CANopen driver or the coupler itself are indicated by the corresponding error class in the PLC (refer to System Technology of the CPU / The diagnosis system in AC500). Furthermore, the CANopen library provides different function blocks which allow detailed error diagnosis (refer to "The CANopen Library").

1.5.4 Function blocks

Libraries:

CANopen_AC500_V11.lib

Group: CAN 2.0A	
CAN2A_INFO	Reading information about CAN 2.0A communication
CAN2A_REC	Reading CAN 2.0A telegrams (with 11 bit identifier) from a receive buffer
CAN2A_SEND	Transmitting CAN 2.0A telegrams (with 11 bit identifier)

Group: CAN 2.0B	
CAN2B_INFO	Reading information about CAN 2.0B communication
CAN2B_REC	Reading CAN 2.0B telegrams (with 29 bit identifier) from a receive buffer
CAN2B_SEND	Transmitting CAN 2.0B telegrams (with 29 bit identifier)

Group: CANopen master / NMT controller	
CANOM_NMT	Controlling NMT node states via network management

Group: CANopen master / Status / Diagnosis	
CANOM_NODE_DIAG	Polling diagnosis data from a slave
CANOM_RES_ERR	Resetting the coupler's error indications
CANOM_STATE	Reading the CANopen coupler status
CANOM_SYS_DIAG	Displaying status surveys of all slaves

Group: SDO parameters	
CANOM_SDO_READ	Reading the value of a slave object
CANOM_SDO_WRITE	Writing the value of a slave object

1.6 Diagnosis

1.6.1 Status LEDs

LED	Color	Status	Meaning
PWR	green	ON	Voltage on
		OFF	Voltage off
RDY	yellow	ON	Coupler ready
		flashes cyclic	Bootstrap loader active
		flashes non-cyclic	Hardware or system error
		OFF	Defective hardware
RUN	green	ON	Communication in progress
		flashes cyclic	Ready for communication
		flashes non-cyclic	Parameterization error
		OFF	No communication or voltage off
STA	yellow	ON	CANopen master: Sends data
		OFF	CANopen master: No data
ERR	red	ON	CANopen error
		OFF	no error

1.6.2 CANopen error messages

The CANopen error messages are listed in the section 'Error messages for the block libraries'.

1.6.3 Function blocks

CANopen master:

Refer to 1.5.4 Function blocks

Online diagnosis:

Refer to the documentation for the field bus configuration tool SYCON.net

1.7 Further information

1.7.1 Standardization

BOSCH CAN specification - version 2.0, part A & part B

ISO 11898

CiA DS 201 V1.1 - CAN Application Layer

CiA DS 301 V3.0 - CAL based Communication Profile for Industrial Systems

CiA DS 301 V4.02 - CANopen Application Layer and Communication Profile

CiA DS 401 V2.1 - CANopen Device Profile Generic I/O modules

CiA DS 402 V2.0 - CANopen Device Profile Driver and Motion Control

CiA DS 406 V3.0 - CANopen Device Profile Encoder

1.7.2 Important address

CAN in Automation (CiA)
Am Weichselgarten 26
D-91058 Erlangen
Germany

Phone: (+49) 9131 69086-0
Fax: (+49) 9131 69086-79
Internet: http://www.can-cia.de

1.7.3 Terms, definitions and abbreviations

CAL	CAN Application Layer
CAN	Controller Area Network
CiA	CAN in Automation international users and manufacturers group e.V.
DLC	Data Length Code
EDS	Electronic Data Sheet
ISO	International Standardization Organization
NMT	Network Management
OD	Object Directory
PDO	Process Data Object
RTR	Remote Transmission Request
SDO	Service Data Object

Index - System Technology of the CANopen Couplers

System Description

AC500

Scalable PLC
for Individual Automation

System Technology
of the DeviceNet Couplers

DeviceNet

Contents "The DeviceNet Coupler"

1 The DeviceNet Coupler

1.1 Brief overview

1.1.1 Fundamental properties and fields of application

DeviceNet is mainly used to transfer process data between central controller modules (such as PLC or PC) and decentralized peripheral modules such as I/O modules, drives and valves. To perform this, DeviceNet uses the physics and data transport mechanisms of the Controller Area Network (CAN). Communication is performed message-oriented. The subscribers exchange data cyclically, change-controlled or event-controlled via the logical interconnections specified during configuration. DeviceNet supports transfer rates of 125 kbaud, 250 kbaud and 500 kbaud.

Each of the up to 64 subscribers in a DeviceNet network has an own MAC ID
(Media Access Control Identifier, bus address). The Duplicate MAC ID test algorithm, which is executed when booting the system, guarantees that each MAC ID exists only once in the network. DeviceNet does not use the conventional master/slave principle. Each subscriber is generally authorized for transmission. As soon as a module reaches a state that activates a configured connection (e.g. changed input value), the module tries to transmit a corresponding telegram via the bus. In this case, it may happen that several subscribers try to send a telegram at the same time. Telegram collisions are detected and removed by the CSMA/CA procedure (Carrier Sense Multiple Access/Collision Avoidance). Each telegram contains a priority identification. The lower this value is, the higher is the priority of the message. The bus address of the sender is part of this priority identification. Due to this, subscribers with a low MAC ID have a higher priority than subscribers with higher addresses when assigning the bus accesses. If a transmission attempt of a subscriber is interrupted by a telegram with a higher priority, the interrupted subscriber disconnects from the bus and then repeats its transmission attempt after the transmission of the high-priority telegram is finished. If a default number of failed transmit attempts is exceeded, the DeviceNet module changes to the fault state. Due to this, normally as low as possible MAC IDs are assigned to DeviceNet masters.

DeviceNet distinguishes three device types: Clients, servers and devices which combine both types of functionality. In principle, all device types can receive (consume) or send (produce) data or perform both. A client (master) typically sends data in form of a request and receives data as an answer to this request. Servers (slaves) typically consume requests and then produce the answers. Depending on the configured connection, slaves are also able to send data via the bus without any previous request by a master. Some devices (clients and servers) can only consume messages (pure output devices) or produce messages (pure input devices).

With DeviceNet, mono master as well as multi master and multi client systems can be realized. In theory each master can access all slaves (servers). The actually realized communication connections are specified during configuration prior to commissioning. Configuration of the connections is done in masters. When booting the system, the master establishes the corresponding connections, i.e. it informs the slaves how to perform communication between the slaves and the master. Pure servers cannot initiate connections, pure clients cannot receive requests. Communication connections are only possible between a master (client) and several slaves (servers) and devices which support both types of functionality. Mixed devices can communicate with other mixed devices as well as with pure servers.

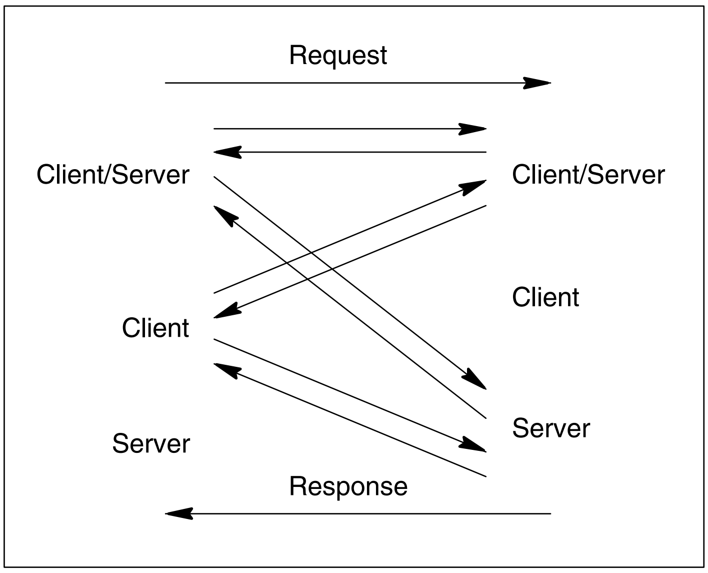


Figure: Communication connections

1.1.2 Communication model

DeviceNet uses the **Producer Consumer Model**. As soon as a DeviceNet subscriber has to transmit data as required by the configured connections, it produces data, i.e. it transmits data via the bus. All modules, which are waiting for data, monitor the bus. If a subscriber recognizes that it is the receiver of the data by reading the identifier contained in the telegram header, it consumes the data, i.e. it reads the complete telegram. Here, it is possible that a message can be directed to one single subscriber (single cast) or several subscribers (multi cast).

DeviceNet defines two types of messages: I/O messaging (I/O data transfer) and explicit messaging (direct connection). I/O messaging is used for exchanging time-critical data (e.g. process data). The data are exchanged via single or multi cast connections and typically use identifiers with a high priority. The meaning of the messages is set with the connection IDs (CAN identifier). Before DeviceNet modules can exchange messages that use these IDs, the modules have to be configured accordingly. The configuration data contain the source and destination address of the object attributes to be transferred for the sender and receiver of the messages. Explicit messaging is a low-priority point to point communication connection between two devices and is typically used for configuration and diagnostic purposes. DeviceNet telegrams (I/O messaging and explicit messaging) contain a maximum of 8 bytes user data. If greater data volumes are to be exchanged, the data have to be split into smaller data packages by fragmenting prior to transmitting. These fragments are then transmitted one after the other and recombined in the receiver.

In addition, DeviceNet knows so-called default master slave connections. These connections use special identifiers of the message group 2. In contrast to all other message types, these types use the MAC ID as destination address and not as source address. Default connections allow to simplify the communication relations. Here, the connection types Bit Strobe, Polling, Change of State and Cyclic are available.

1.1.3 Connection types

For Change of State connections (transfer in case of a state change), a DeviceNet subscriber only transmits the data if they have changed since the last transmission. In case of slow processes, this can lead to longer send pauses. Due to this, all devices are equipped with an adjustable heartbeat timer. If this heartbeat timer is elapsed and the data did not change in the mean time, they are transmitted nevertheless. The heartbeat timer is triggered with every new transmit process. That means, the subscriber transmits the data in case of a state change, but definitely when the heartbeat interval is elapsed. This way, the receiver is informed that the sender works correctly and did not fail. Hence, transmitting a request to the sender of the data is not required.

Cyclic connections are mainly used if a fixed time slot pattern is set by default. If, for example, a temperature sensor is used in a slow closed-loop control circuit with a sampling rate of 500ms, it is recommended to update the temperature value cyclically in intervals of 500ms. Shorter update intervals would not provide additional information to the loop controller but cause unnecessary high load on the bus instead.

Change of State connections and Cyclic connections are Acknowledged Exchanges per default, i.e. the consumer (receiver) of the data transmits an acknowledgement to the producer (transmitter) as a receipt confirmation. These acknowledgements can be suppressed using the Acknowledge Handler Object (refer to the object model) in order to avoid additional telegram traffic for systems which are anyway heavily loaded due to fast update times.

Multi Cast connections are also managed using the Acknowledge Handler Object. The slave data are not only read by the master but also by other subscribers, e.g. by operating stations which monitor the telegram traffic and process and acknowledge the data they require for display, alarm and archive purposes.

In case of Polling connections, the master transmits a poll command to the slave. If the master contains data intended for the slave, these data are also transferred. This can be performed using one single telegram or with help of the fragmentation services. Data volumes with a size greater than 8 bytes are divided into subsets and then transferred one after the other. The slave recombines the data packages received from the master. If the slave contains data that are directed to the polling master, it transmits these data to the master. The response telegram of the slave alternatively or additionally contains status information. If the slave does not respond to a polling request of the master, a timeout occurs. Especially in case of slow processes, a disadvantage of Polling connections is that they cause unnecessary high load on the bus since often the data do not change between two procedures.

Bit Strobe connections are used to transfer small data volumes between a master and one or several slaves. The Bit Strobe telegram transmitted by the master contains 64 bits of user data. Each bit is assigned to one subscriber or bus address. Therefore, Bit Strobe telegrams have a broadcast functionality. As all addressed slaves receive the telegram at the same time, Bit Strobe is often used for the synchronization of input or output data. A slave can provide the assigned bit directly at an output or interpret the bit as a trigger condition in order to freeze input values and to transfer these values to the master with the next polling telegram. The data sent back by the slave are limited to a size of 8 bytes. Thus, Bit Strobe causes lower bus load than the Polling method.

1.1.4 Object model

DeviceNet is based on an abstract object model. This model defines the organization and implementation of the attributes (data), services (methods, procedures) and the behavior of the individual components of a DeviceNet subscriber. The complete definition is part of the corresponding specification that can be obtained from the ODVA. The model provides a four-step addressing schema for each attribute. The first level is the NodeAddress (MAC ID, bus address), the second is the Object Class Identifier, the third level is the Instance Number and the last level is the Attribute Number.

Classes:

Each DeviceNet module contains a collection of objects where each object has a certain class. The DeviceNet standard already contains the definition of many standard classes. These standard classes describe, for example, fundamental properties, the communication behavior or parameters of individual channels of a subscriber. In addition, further manufacturer-specific classes can be defined for a DeviceNet module.

The definition of classes can be compared to the definition of STRUCT data types or function blocks included in libraries. When defining a structure, no data are created at first. Only a possible data format is described. If a function block library is inserted into a project, the included function blocks are not executed. The library only contains descriptions of the available function blocks.

Objects and instances:

Each DeviceNet subscriber contains one or several objects of the different classes. An object is an instance of a class. Depending on the type of class, only one object or several objects of this class can be available. The instances are numbered continuously.

This is comparable with the creation of variables of previously defined STRUCT types. For a library, this corresponds to the insertion of a function block included in the library into the project. After a symbolic name has been entered, an instance of this type is generated in the project.

Attributes:

Data (attributes) are combined in an object instance. These attributes describe the different properties of a DeviceNet module and can be (partly) read or written by other modules via the bus.

In relation to the STRUCT variables in a project, attributes can be compared with the individual elements of a structure. Reading and writing the individual elements is done by assignments in the program. For function blocks, the inputs and outputs can be seen as attributes.

Standard objects:

The standard objects described below are contained in any DeviceNet module, provided no other information is available in this document. Some information contained in these objects are, for example, read automatically by a DeviceNet master from the particular device when commissioning the bus and are then compared with the required configuration. When establishing the communication connections during the boot process of the system, a master writes objects to the assigned slaves. This informs the slave how it has to use the bus.

Each DeviceNet module has an instance of the Identity Object (class identifier 1, instance number 1). Parts of this instance are, for example, the attributes manufacturer code, device type, product code, version identifier, status, serial number and product name. This information can be read out from a device using the DeviceNet service Get_Attribute_Single. The following table shows the structure of the Identity Object.

Identity Object: class identifier 1, instance number 1				
Attribute No.	**Attribute name**	**Data format**	**Description**	**Implementation**
1	Vendor ID	UINT	Manufacturer identifier	required
2	Device Type	UINT	Device type	required
3	Product Code	UINT	Product number	required
4	Major Revision Minor Revision	USINT USINT	Version identifier	required
5	Status	WORD	General status	required
6	Serial Number	UDINT	Serial number	required
7	Product Name	STRING(32)	Product name	required
8	State	USINT	Current state	optional
9	Configuration Consistency Value	UINT	Device configuration	optional
10	Heartbeat Interval	USINT	Time interval in seconds	optional

Table: Structure of the Identity Object

In addition, DeviceNet modules have an instance of the Message Router Object. The Message Router Object is part of a device which explicitly routes messages to other objects. Usually it is not possible to directly access this object within the DeviceNet network.

The instance of a DeviceNet object of a device contains the attributes (data) bus address (MAC ID), baud rate, bus off reaction, bus off counter, allocation selection and the MAC ID of the master. These data are also accessed using the Get_Attribute_Single service.

DeviceNet modules have at least one Assembly Object. The main task of these objects is to combine different attributes (data) of different Application Objects into one attribute that can be transmitted in one single message.

In addition, DeviceNet modules have at least two Connection Objects. Each Connection Object represents one end point of a virtual connection between two subscribers of a DeviceNet network. These virtual connections are named explicit messaging and I/O messaging (refer to the description above). Explicit messages contain the address and value of an attribute as well as a service identifier that describes how the data are handled. In contrast, an I/O message contains only data. The information how the data have to be handled are contained in the Connection Object to which this message is assigned.

The Parameter Object is an optional object and is only used by devices which have adjustable parameters. An own instance is available for each parameter. The Parameter Object provides standardized access to all parameters for the configuration tools. The configuration options are attributes of the Parameter Object and can be, for example, a value range or scaling of channels, texts or limits.

Normally, at least one Application Object exists in each DeviceNet module in addition to the objects of the assembly or parameter class. At this point, these objects are not described in detail. A description can be found in the DeviceNet Object library.

1.1.5 EDS files

The characteristic properties of a DeviceNet module are documented in the form of an electronic data sheet (EDS file). The file completely and clearly describes the characteristics (objects) of a module type in a standardized and manufacturer-independent format. Programs used to configure a DeviceNet network use the module type descriptions available in the EDS files. This strongly simplifies the configuration of a DeviceNet system. Usually the EDS files are provided by the device manufacturer. In addition, the ODVA provides many EDS files in the Internet which can be downloaded free of charge.

The Internet address of the ODVA is: http://www.odva.org.

1.1.6 Features

Mode of operation:

- DeviceNet master (client)
- Check for double subscriber addresses (MAC IDs) on the bus when initializing the system
- A maximum of 256 bytes of input data and 256 bytes of output data per slave
- A maximum of 57344 I/O points

Transmission technique:

- ISO 11898, potential separated
- Data lines, twisted-pair line and power supply in only one cable
- Transfer rates of 125 kbit/s, 250 kbit/s and 500 kbit/s
- Bus length up to 500 m at 125 kbit/s and up to 100 m at 500 kbit/s (Trunk Cable)
- One bus can have up to 64 subscribers
- 5-pin COMBICON socket for bus connection

Communication:

- Message-oriented bus access, CSMA/CA
- Data transfer: Poll, bit strobe, cyclic or change of state, explicit peer-to-peer (acyclic) via class 2, UCMM group 1, 2 and 3 fragmentation
- Predefined master-slave connections
- 8 bytes of non-fragmented user data, for fragmentation any size is possible
- Synchronization of inputs and/or outputs via bit strobe connections

Protection functions:

- Message transfer with Hamming distance HD = 6
- Fault recognition mechanisms via 15 bit CRC, frame check, acknowledge, bit monitoring and bit stuffing
- Incorrect parameter settings are avoided because bus subscribers with faulty parameters are not included in the user data operation
- The system keeps running in case of a subscriber failure. The failure is registered in the master and indicated via a common diagnosis.
- Response monitoring of the subscribers (heartbeat) by a cyclic check for double subscriber addresses
- Removing, adding and exchanging of subscribers during running operation is possible

Status indication with 4 LEDs:

- PWR (green): Power supply for the coupler
- RDY (yellow): Coupler ready for operation
- RUN (green): Configuration and communication status
- NET/MOD (green/red): Module status

1.2 Technical data

1.2.1 Technical data of the coupler

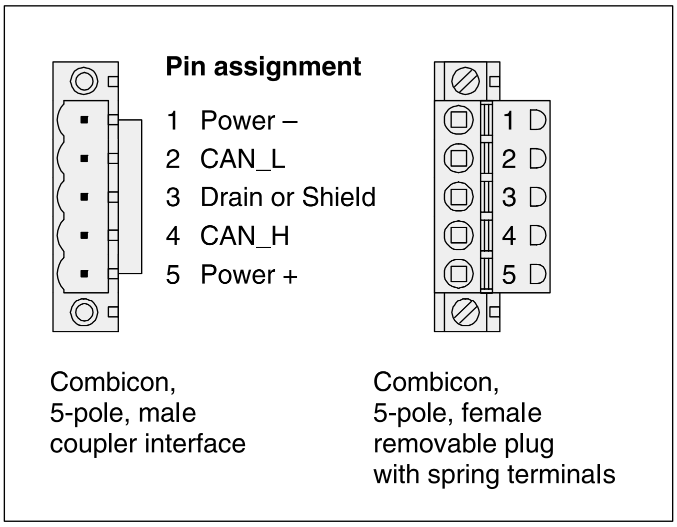
Coupler CM575-DN	
Field bus	DeviceNet
Transmission rate	125 kbit/s up to 500 kbit/s
Protocol	DeviceNet master
Attachment plug for field bus	Pluggable 5-pin COMBICON connector
Processor	EC1, 160 pins
Clock frequency	48 MHz
Possible CPUs	PM571-xxx, PM581-xxx, PM591-xxx
Possible terminal bases	All
Ambient temperature	0 °C...55 °C
Coupler interface	Dual-port memory, 8 kbytes
Current consumption over the coupler bus	typ. 180 mA
Internal RAM memory (EC1)	256 kbytes
External RAM memory	-
External Flash memory	512 kbytes (firmware)
Status display	PWR, RDY, RUN, NET, MOD
Weight	approx. 150 g

Table: Technical data of the DeviceNet coupler

1.2.2 Technical data of the interface

Interface socket	5-pin COMBICON
Transmission standard	ISO 11898, potential-free
Transmission protocol	DeviceNet, 500 kBaud max.
Transmission rate	Baud rate 125 kbit/s, 250 kbit/s, 500kbit/s
Status indication	4 LEDs
Number of subscribers	63 max.

The pin assignment of the DeviceNet interface is as follows:

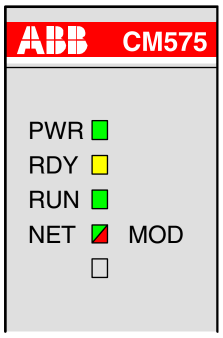

1
2
3
4
5
Combicon, 5-pole, male coupler interface
Combicon, 5-pole, female removable plug with spring terminals

Pin assignment

1	Power –
2	CAN_L
3	Drain or Shield
4	CAN_H
5	Power +

Figure: Pin assignment of the field bus interface DeviceNet

The following figure shows the position of the LEDs. The LED states and their meanings are described in the following table.



LED	Color	State	Meaning
PWR	green	ON	Voltage on
		OFF	Voltage off
RDY	yellow	ON	Coupler ready
		flashes cyclic	Bootstrap loader active
		flashes non-cyclic	Hardware or system error
		OFF	Defective hardware or voltage off
RUN	green	ON	Communication in progress
		flashes cyclic	Ready for communication
		flashes non-cyclic	Parameterization error
		OFF	No communication or voltage off
NET/ MOD	green/ red	ON (green)	Device is in online mode and has established one or several connections
		flashes cyclic (green)	Device is in online mode and has established no connection
		flashes cyclic (green/red)	Faulty communication
		ON (red)	Critical connection error, device has detected a network error (duplicated MAC ID or bus interrupted)
		flashes cyclic (red)	Connection timeout
		OFF	After start of the device and during duplicate MAC ID check

1.3 Connection and data transfer media

1.3.1 Attachment plug for the bus cable

Assignment:

5-pin COMBICON connector

Pin no. of the interface plug (from top to bottom)	Signal	Typical core color	Meaning
1	Power –	black	Reference potential for external power supply
2	CAN_L	blue	Receive/transmit line LOW
3	Drain	blank	Shield of the bus cable
4	CAN_H	white	Receive/transmit line HIGH
5	Power +	red	+24 V, external power supply

Table: Pin assignment of the attachment plug for the bus cable

It is absolutely necessary that all cables (i.e. the data lines CANH / CANL, the external 24 V power supply +V / -V and the shielding) are connected.

Supplier:

e.g. COMBICON

Phoenix Contact GmbH & Co.
Flachsmarktstraße 8 - 28
32825 Blomberg, Germany

Phone: (+49) (0)52 35 / 3-00
Fax: (+49) (0)52 35 / 3-4 12 00
Internet: http://www.phoenixcontact.com

1.3.2 Bus terminating resistors

The ends of the data lines (bus segments) have to be terminated with a 120Ω bus termination resistor. The bus termination resistor is usually installed directly at the bus connector.

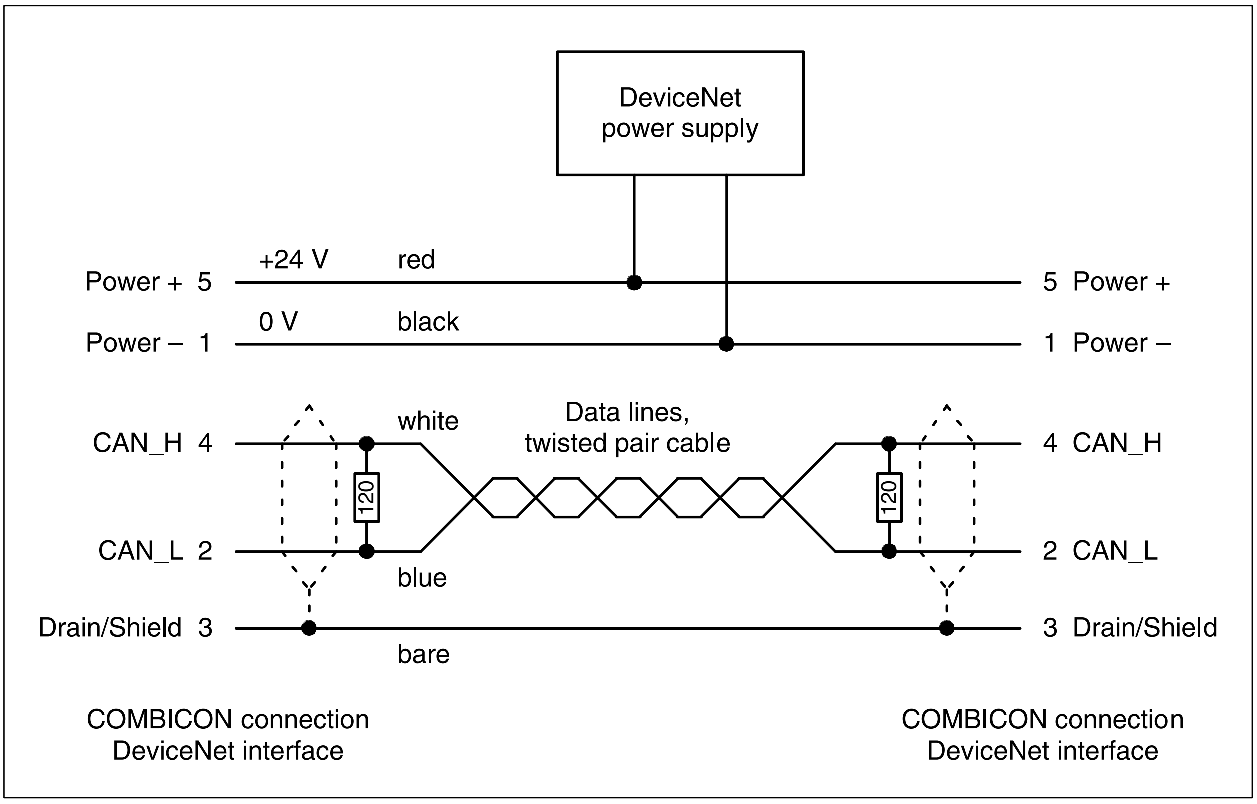


Figure: DeviceNet interface, bus terminating resistors connected to the line ends

1.3.3 Bus cables

Two cable types are used with DeviceNet: Trunk Cables and Drop Cables. The Trunk Cable has a greater line diameter than the Drop Cable. In principle, both line types can be used as main lines (Trunk Line) and as branch lines (Drop Line). However, using the thinner line reduces the maximum line lengths.

Trunk Cable (main lines):

Data lines:

Type	Twisted pair cable (shielded)
Wave impedance (cable impedance)	120 Ω
Cable capacity (distributed capacitance)	≤ 40 nF/km
Conductor cross section	≥ 1.0 mm² (18 AWG / 19)
Line resistance per core	≤ 22.5 Ω/km
Loop resistance (serial resistance of 2 cores)	≤ 45 Ω/km

Power supply:

Loop resistance (serial resistance of 2 cores)	≤ 45 Ω/km
Conductor cross section	≥ 1.5 mm² (15 AWG / 19)

Drop Cable (branch lines):

Data lines:

Type	Twisted pair cable (shielded)
Wave impedance (cable impedance)	120 Ω
Cable capacity (distributed capacitance)	≤ 40 nF/km
Conductor cross section	≥ 0.25 mm² (24 AWG / 19)
Line resistance per core	≤ 92 Ω/km
Loop resistance (serial resistance of 2 cores)	≤ 184 Ω/km

Power supply:

Loop resistance (serial resistance of 2 cores)	≤ 45 Ω/km
Conductor cross section	≥ 0,34 mm² (22 AWG / 19)

Supplier:

e.g. UNITRONIC® BUS DeviceNet

U.I. LAPP GmbH
Schulze-Delitzsch-Straße 25
70565 Stuttgart, Germany
Phone: (+49) (0)711 7838 01
Fax: (+49) (0)711 7838 264
Internet: http://www.lappkabel.de

1.3.4 Maximum line lengths

Transmission rate	125 kbit/s	250 kbit/s	500 kbit/s
Max. line length main strand Trunk Cable	500 m (1.610 ft)	250 m (820 ft)	100 m (328 ft)
Max. line length main strand Drop Cable	100 m (328 ft)	100 m (328 ft)	100 m (328 ft)
Max. line length per branch line Trunk Cable / Drop Cable	6 m (20 ft)	6 m (20 ft)	6 m (20 ft)
Max. line length: sum of branch lines Trunk Cable / Drop Cable	156 m (512 ft)	78 m (256 ft)	39 m (128 ft)

Table: Maximum line lengths

1.4 Possibilities for networking

The DeviceNet coupler is connected to the bus using the 5-pin COMBICON socket. For EMC suppression and protection against dangerous contact voltages, the shield of the bus line has to be connected to protective earth outside the housing.

1.4.1 Single master system

The single master system is the simplest version of a DeviceNet network. It consists of a DeviceNet master (Client) and up to 63 slaves (Server). The line ends of the busses have to be terminated using bus termination resistors.

The DeviceNet master is able to:

- automatically determine the structure of the system, i.e. the description and configuration possibilities of all subscribers.
- detect double bus addresses in the network.
- parameterize the slaves (e.g. communication connections, time supervision, bus traffic).
- configure slaves (e.g. type, number and channel operating mode).
- read input data of the slaves.
- write output data of the slaves.
- read diagnostic data of the slaves.
- send control commands to the slaves (via bit strobe, e.g. freeze input signals).
- read, write and reset slave objects even if the slaves are in running mode.

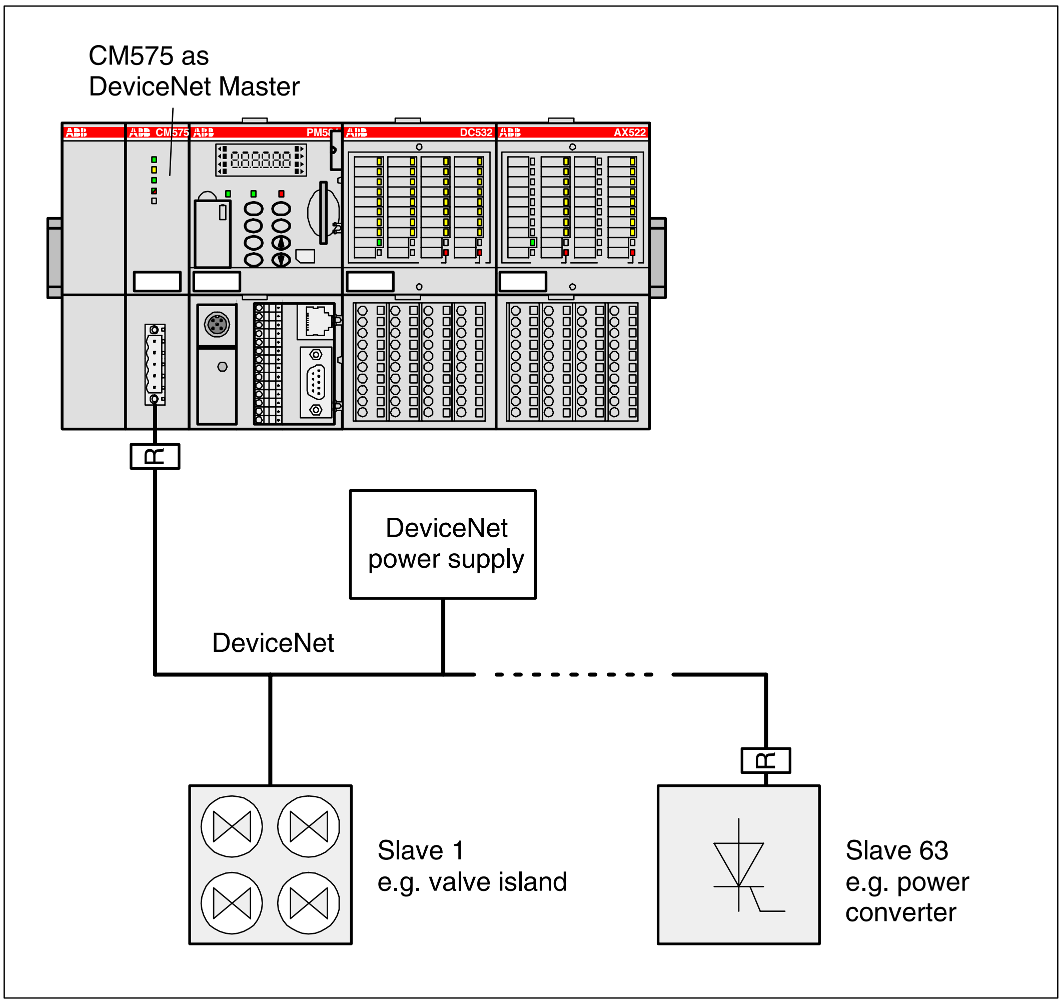


Figure: Example of a single master system

1.4.2 Multi master system

A DeviceNet network that contains several masters is called a multi master system. Up to 64 subscribers (master and slaves) can be operated in this type of network. In a multi master system, master-slave communication is possible as well as the data exchange between masters.

Masters are able to:

- automatically determine the structure of the system, i.e. the description and configuration possibilities of all subscribers.
- parameterize the slaves (e.g. communication connections, time supervision, bus traffic).
- configure slaves (e.g. type, number and channel operating mode).
- read input data of the slaves.
- write output data of the slaves.
- read diagnostic data of the slaves.
- send control commands to the slaves (via bit strobe, e.g. freeze input signals).
- read, write and reset slave objects even if the slaves are in running mode.

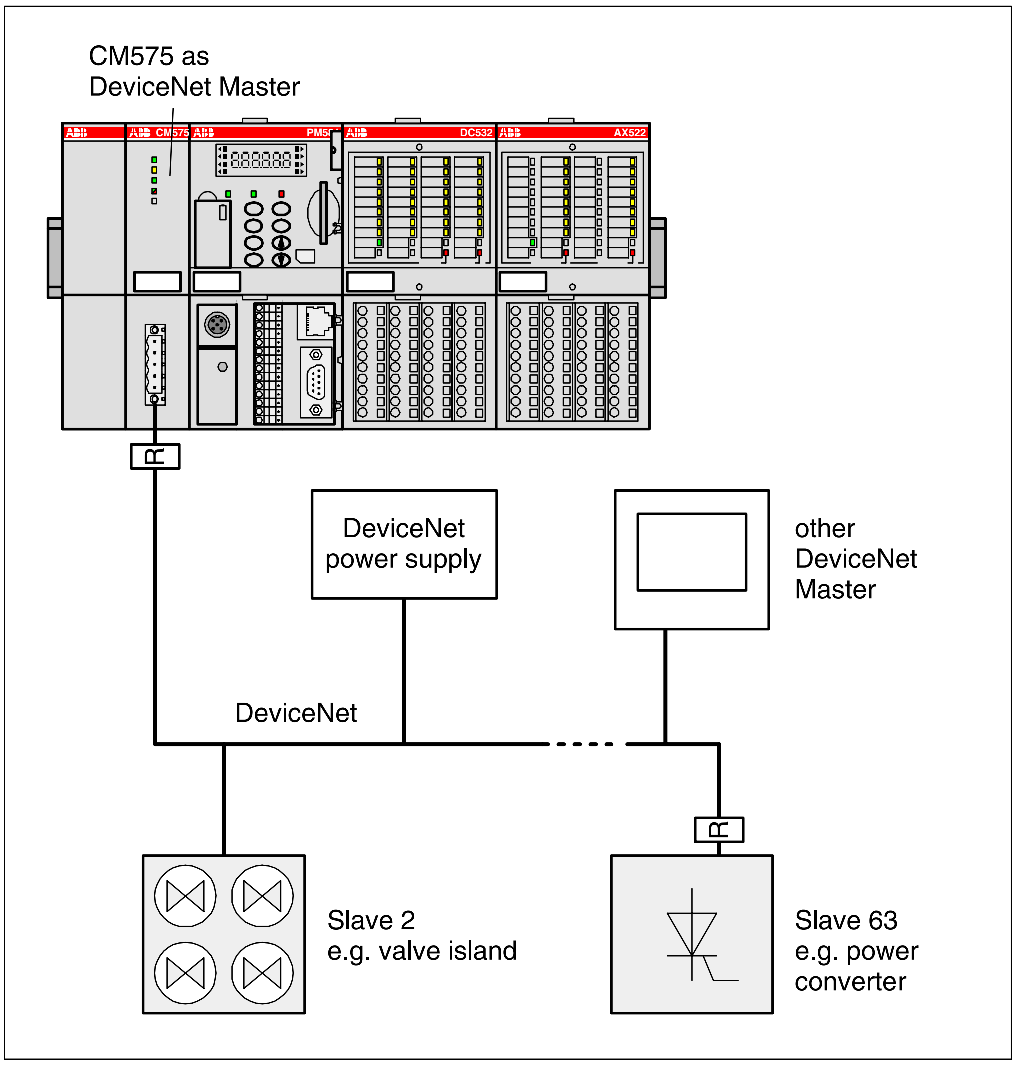


Figure: Example of a multi master system

1.5 DeviceNet implementation

1.5.1 Configuration

The integration of the coupler into the PLC configuration of the AC500 is an assumption for the correct function of the DeviceNet coupler CM575. The configuration of the coupler and the connected DeviceNet subscribers is done using the tool SYCON.net which is part of the Control Builder programming software.

In the following configuration example, the coupler CM575 is configured as DeviceNet master device. The functionality of the DeviceNet slave is made available via an FBP DeviceNet plug at a motor controller (UMC).

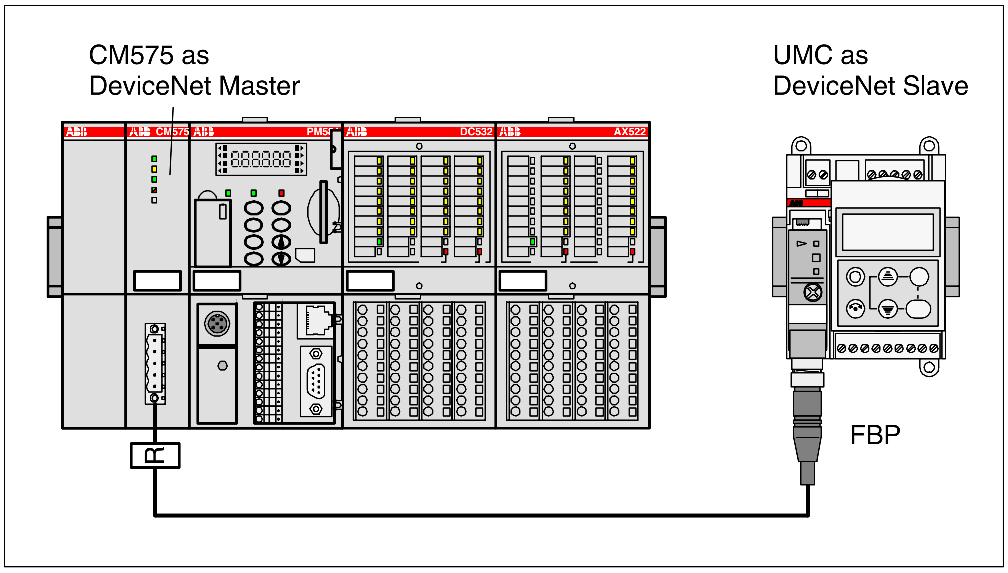
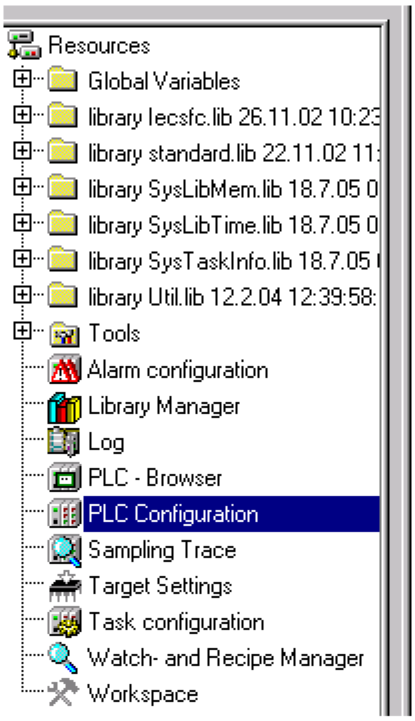
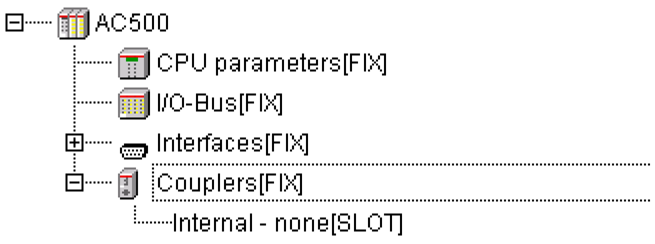


Figure: Example configuration consisting of an AC500 with a DeviceNet coupler and a DeviceNet slave (FBP on a UMC)

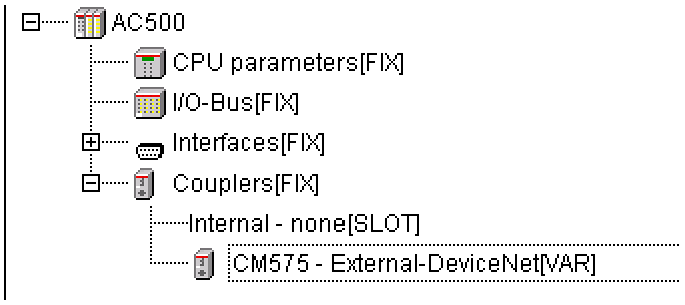
PLC configuration

The configuration of the DeviceNet coupler starts with the integration of the coupler into the PLC configuration.

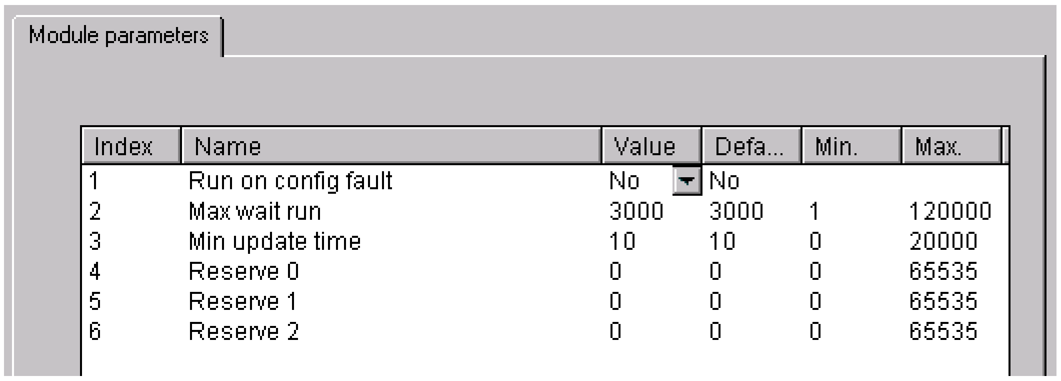

Resources
Global Variables
library Iecsfc.lib 26.11.02 10:23
library standard.lib 22.11.02 11:
library SysLibMem.lib 18.7.05 0
library SysLibTime.lib 18.7.05 0
library SysTaskInfo.lib 18.7.05
library Util.lib 12.2.04 12:39:58:
Tools
Alarm configuration
Library Manager
Log
PLC - Browser
PLC Configuration
Sampling Trace
Target Settings
Task configuration
Watch- and Recipe Manager
Workspace


AC500
CPU parameters[FIX]
I/O-Bus[FIX]
Interfaces[FIX]
Couplers[FIX]
Internal - none[SLOT]

To insert the coupler into the configuration, select "Couplers", press the right mouse button and then select "Append Subelement" -> "CM575".


```
AC500
  CPU parameters[FIX]
  I/O-Bus[FIX]
  Interfaces[FIX]
  Couplers[FIX]
    Internal - none[SLOT]
    CM575 - External-DeviceNet[VAR]
```

Do not change the default values for the coupler parameters.

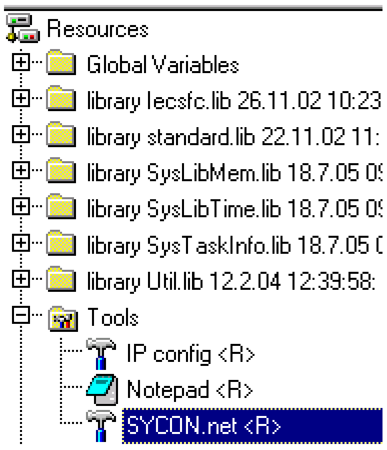

Module parameters

Index	Name	Value	Defa...	Min.	Max.
1	Run on config fault	No	No		
2	Max wait run	3000	3000	1	120000
3	Min update time	10	10	0	20000
4	Reserve 0	0	0	0	65535
5	Reserve 1	0	0	0	65535
6	Reserve 2	0	0	0	65535

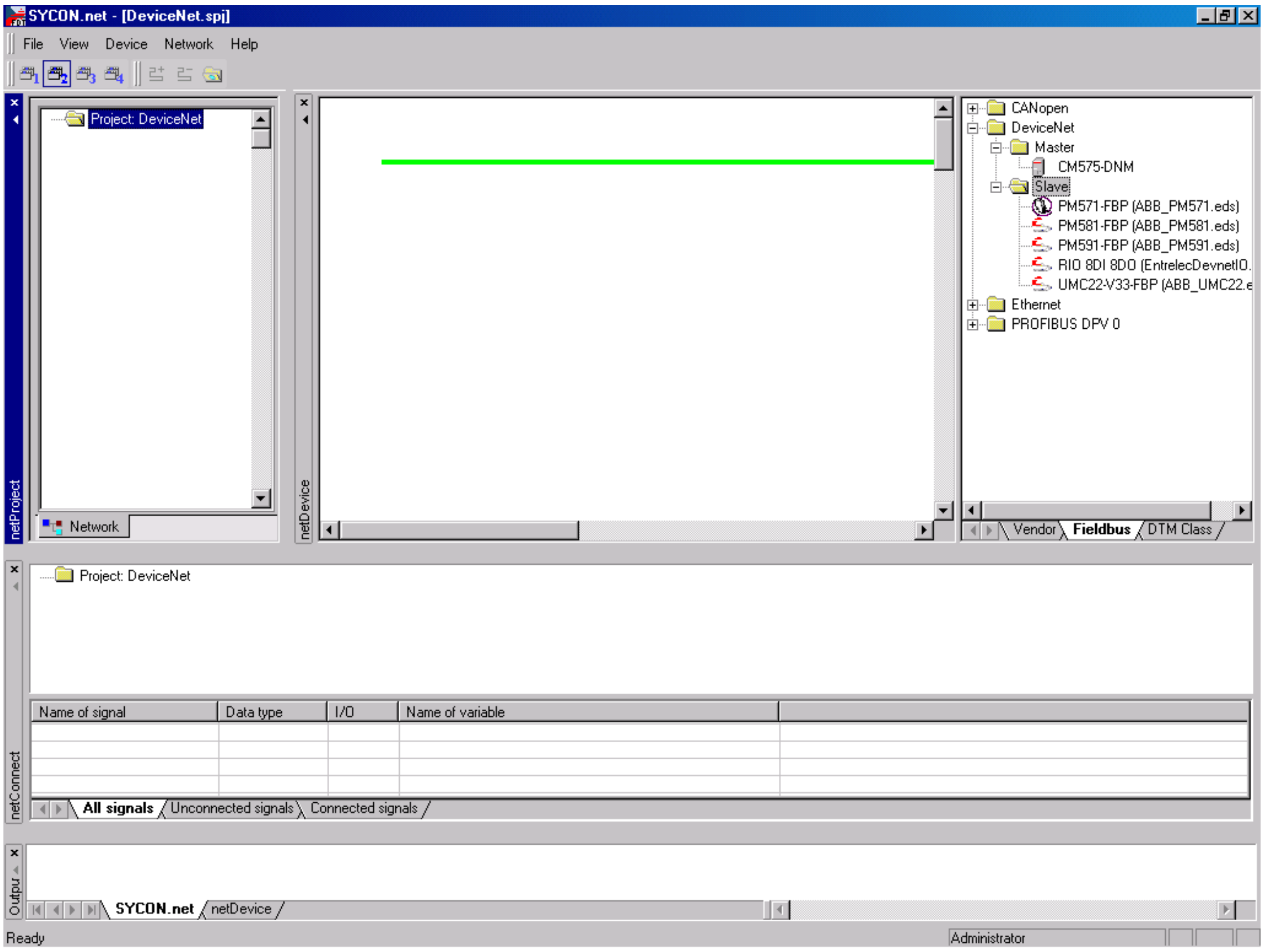
The coupler is now integrated in the PLC configuration.

Configuration using SYCON.net

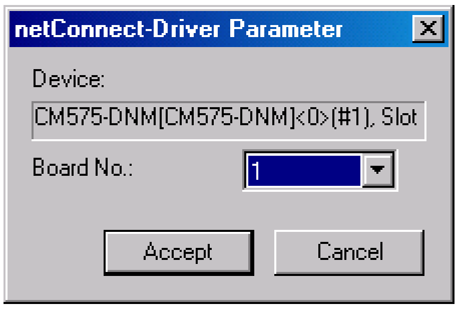
When configuring the DeviceNet coupler, the configuration data are a definite element of a project. They are specified using the tool SYCON.net (Resources tab -> Tools -> SYCON.net) which is part of the Control Builder. The configuration data are transferred to the coupler with the SYCON.net tool.


```
Resources
  Global Variables
  library Iecsfc.lib 26.11.02 10:23
  library standard.lib 22.11.02 11:
  library SysLibMem.lib 18.7.05 0
  library SysLibTime.lib 18.7.05 0
  library SysTaskInfo.lib 18.7.05 (
  library Util.lib 12.2.04 12:39:58:
  Tools
    IP config <R>
    Notepad <R>
    SYCON.net <R>
```

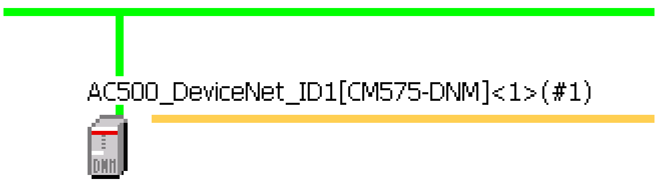
The following view appears when starting the configuration tool SYCON.net.


SYCON.net - [DeviceNet.spj]
File View Device Network Help
Project: DeviceNet
Network
CANopen
DeviceNet
Master
CM575-DNM
Slave
PM571-FBP (ABB_PM571.eds)
PM581-FBP (ABB_PM581.eds)
PM591-FBP (ABB_PM591.eds)
RIO 8DI 8DO (EntrelecDevnetIO.
UMC22-V33-FBP (ABB_UMC22.e
Ethernet
PROFIBUS DPV 0
Vendor Fieldbus DTM Class
netProject
netDevice
Project: DeviceNet
Name of signal
Data type
I/O
Name of variable
All signals Unconnected signals Connected signals
netConnect
SYCON.net netDevice
Outpu
Ready
Administrator

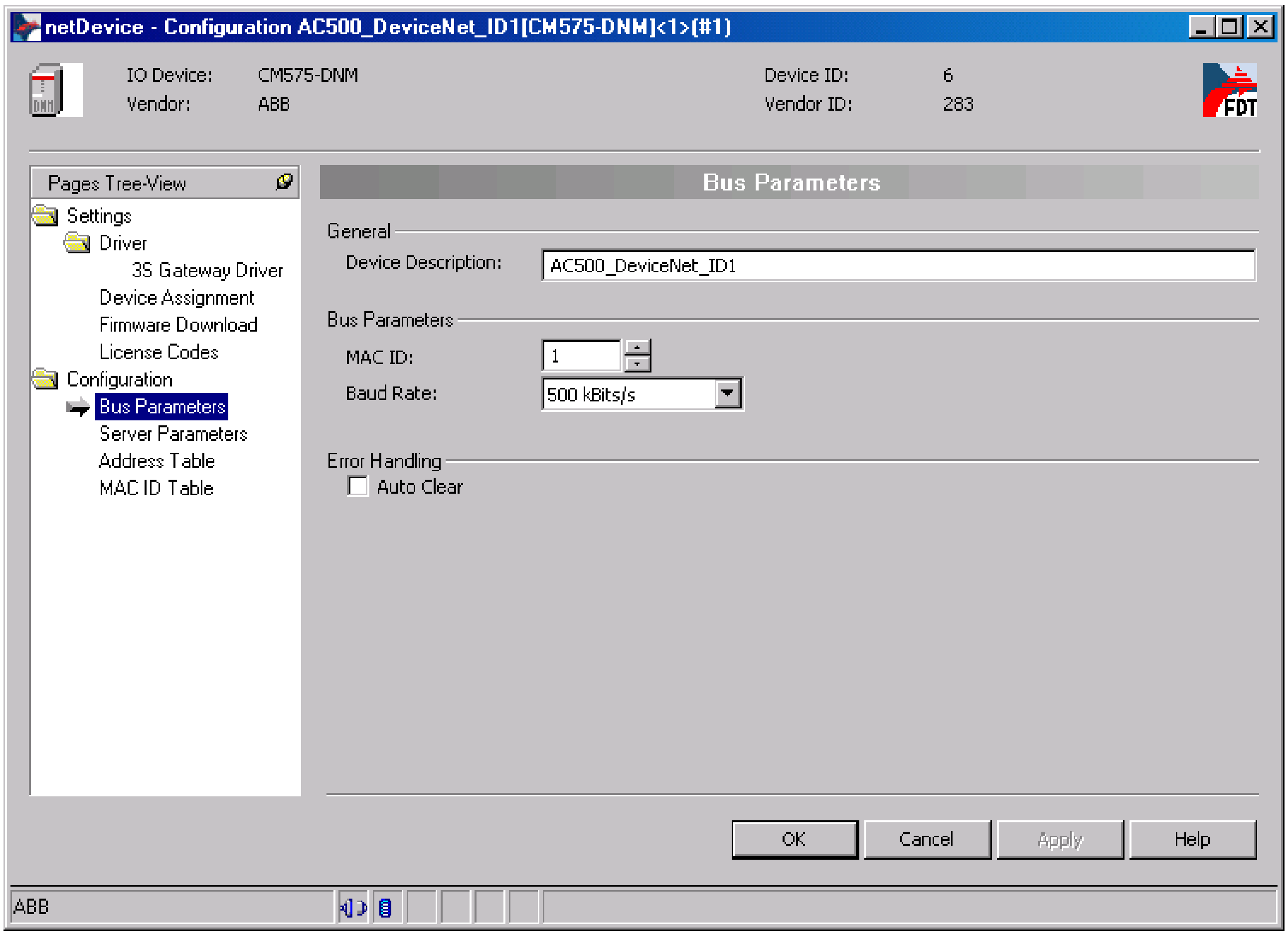
In the top right window, click on the entry "CM575-DNM" in the folder "DeviceNet/Master" and drag it onto the green line displayed in the middle window. Correct insertion positions are displayed by a "+".


netConnect-Driver Parameter
Device:
CM575-DNM[CM575-DNM]<0>(#1), Slot
Board No.:
1
Accept
Cancel

A dialog appears where you have to select the board number according to the coupler slot. The first slot left of the CPU is slot 1 (or board number 1).



To configure the DeviceNet master, place the cursor on the "CM575" icon and then press the right mouse button and select "Configuration".


netDevice - Configuration AC500_DeviceNet_ID1[CM575-DNM]<1>(#1)

IO Device: CM575-DNM
Vendor: ABB
Device ID: 6
Vendor ID: 283

Bus Parameters

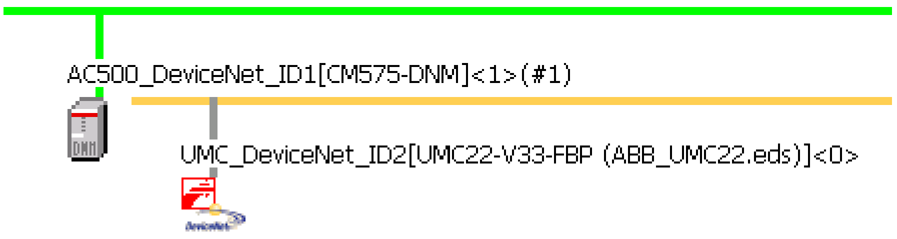
General
Device Description: AC500_DeviceNet_ID1

Bus Parameters
MAC ID: 1
Baud Rate: 500 kBits/s

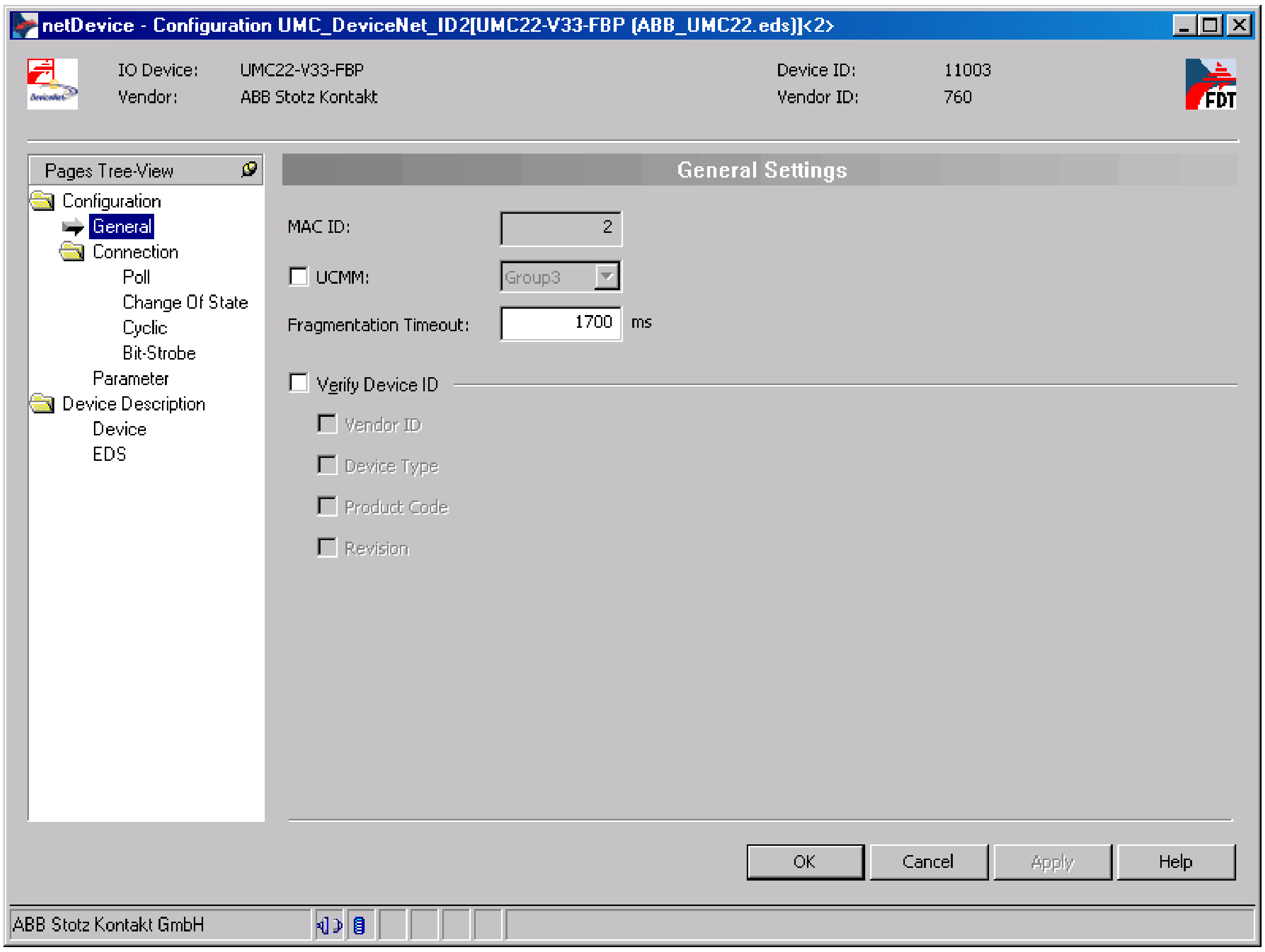
Error Handling
☐ Auto Clear

Enter the MAC ID, select the baud rate of the DeviceNet coupler and apply the parameters by pressing "OK".

In the top right window, click on the entry "UMC22-V33-FBP" in the folder "DeviceNet/Slave" and drag it onto the yellow line (DeviceNet line) displayed in the middle window. Correct insertion positions are displayed by a "+".


AC500_DeviceNet_ID1[CM575-DNM]<1>(#1)
UMC_DeviceNet_ID2[UMC22-V33-FBP (ABB_UMC22.eds)]<0>

To configure the DeviceNet slave, place the cursor on the slave icon and then press the right mouse button and select "Configuration".



netDevice - Configuration UMC_DeviceNet_ID2[UMC22-V33-FBP (ABB_UMC22.eds)]<2>

IO Device: UMC22-V33-FBP
Vendor: ABB Stotz Kontakt
Device ID: 11003
Vendor ID: 760

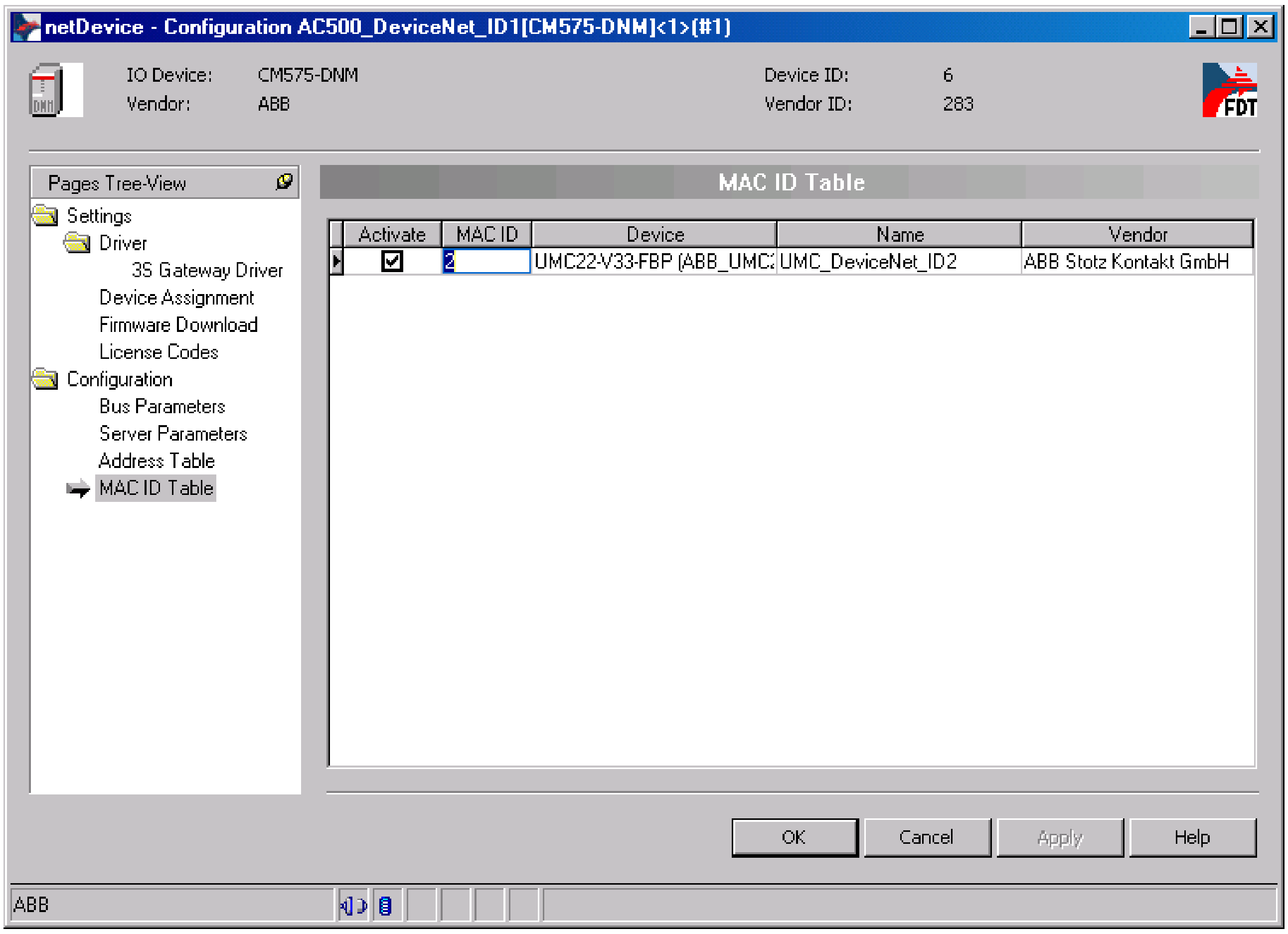
Pages Tree-View
- Configuration
  - General
  - Connection
    - Poll
    - Change Of State
    - Cyclic
    - Bit-Strobe
  - Parameter
- Device Description
  - Device
  - EDS

General Settings

MAC ID: 2
☐ UCMM: Group3
Fragmentation Timeout: 1700 ms

☐ Verify Device ID
- ☐ Vendor ID
- ☐ Device Type
- ☐ Product Code
- ☐ Revision

In this example, we do not change the configuration of the DeviceNet slave. Only the MAC ID of the slave is changed according to the HW setting. This is done in the master configuration dialog. Open the master configuration and select the subitem "MAC ID table" in the "Configuration" folder.



netDevice - Configuration AC500_DeviceNet_ID1[CM575-DNM]<1>(#1)

IO Device: CM575-DNM
Vendor: ABB
Device ID: 6
Vendor ID: 283

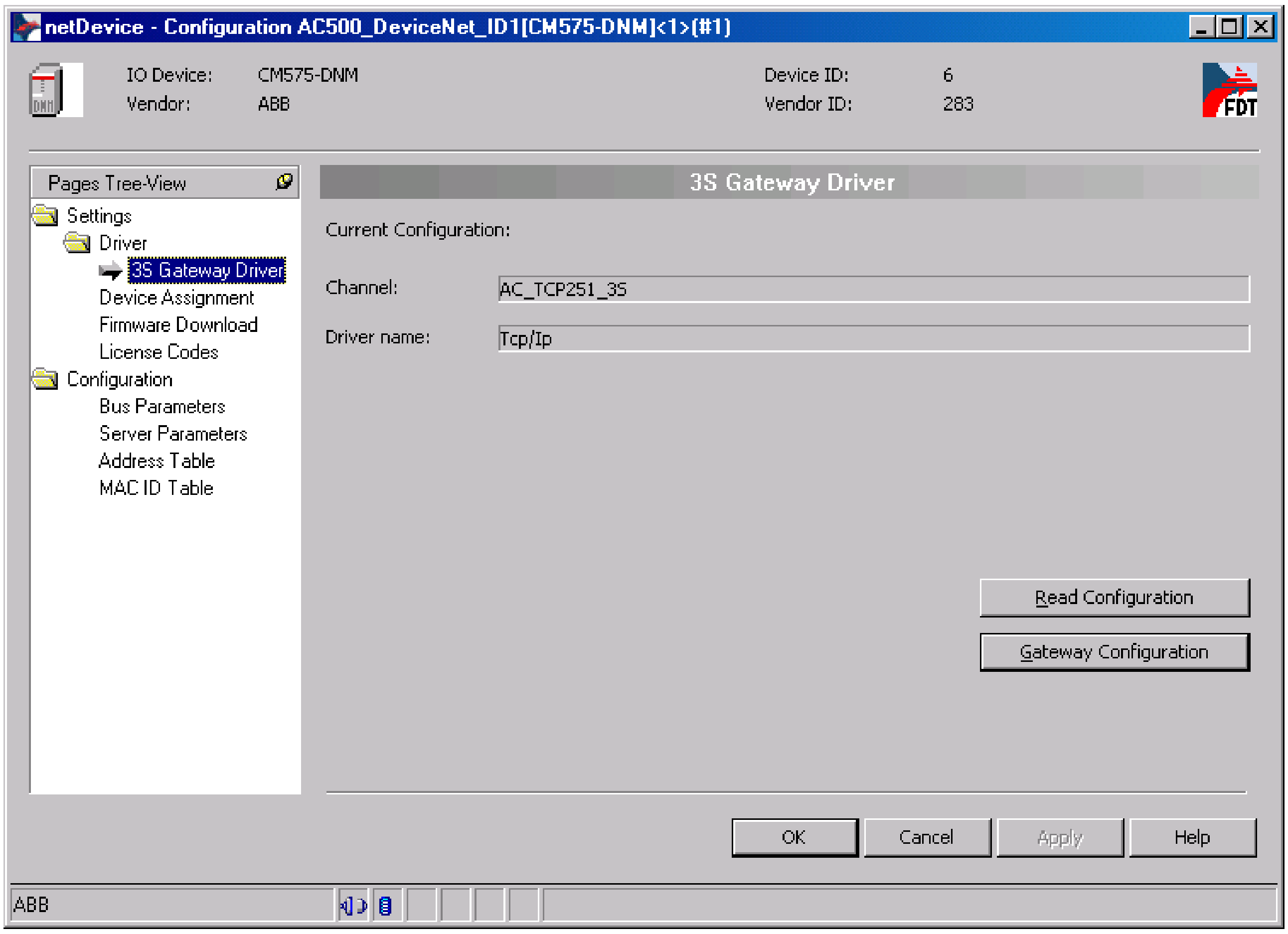
MAC ID Table

Activate	MAC ID	Device	Name	Vendor
☑	2	UMC22-V33-FBP (ABB_UMC2	UMC_DeviceNet_ID2	ABB Stotz Kontakt GmbH

ABB

Set the MAC ID of the slave. Leave all other settings unchanged.

Now the configuration can be downloaded to the DeviceNet master. To do this, first the interface has to be specified that is used to download the data to the coupler. The interface to be used is configured in the tab "Settings -> Driver -> 3S Gateway Driver" in the DeviceNet master configuration.


netDevice - Configuration AC500_DeviceNet_ID1[CM575-DNM]<1>(#1)

IO Device: CM575-DNM
Vendor: ABB
Device ID: 6
Vendor ID: 283

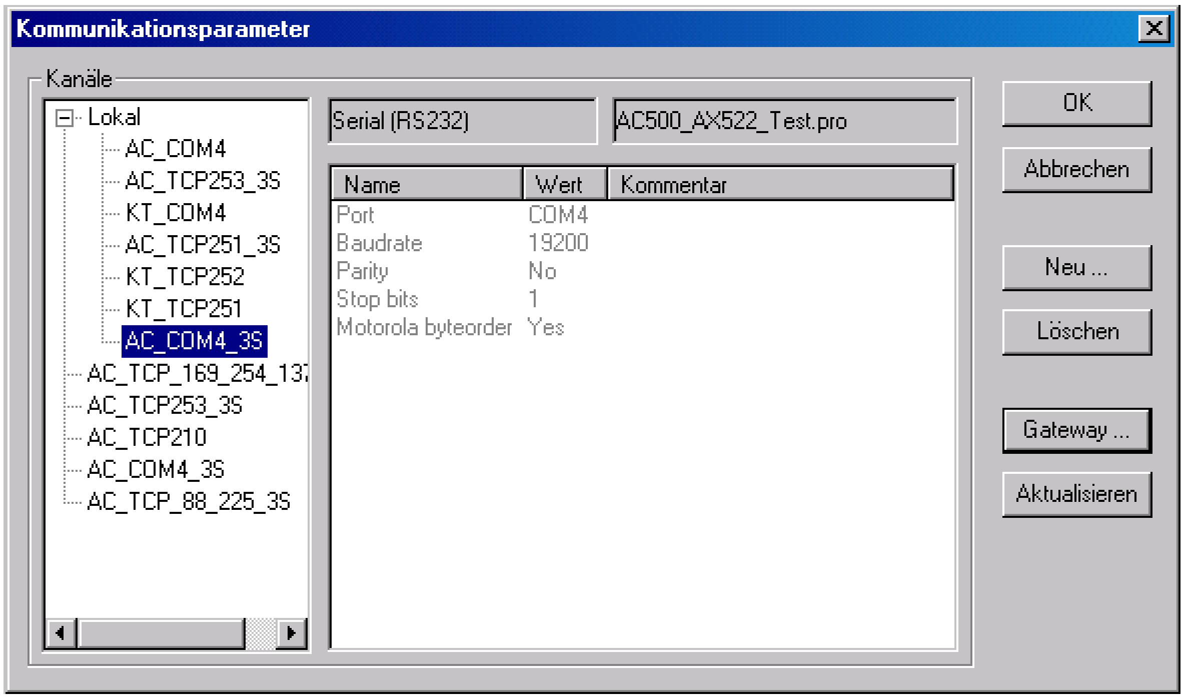
3S Gateway Driver

Current Configuration:

Channel: AC_TCP251_3S

Driver name: Tcp/Ip

Press the button "Gateway Configuration".

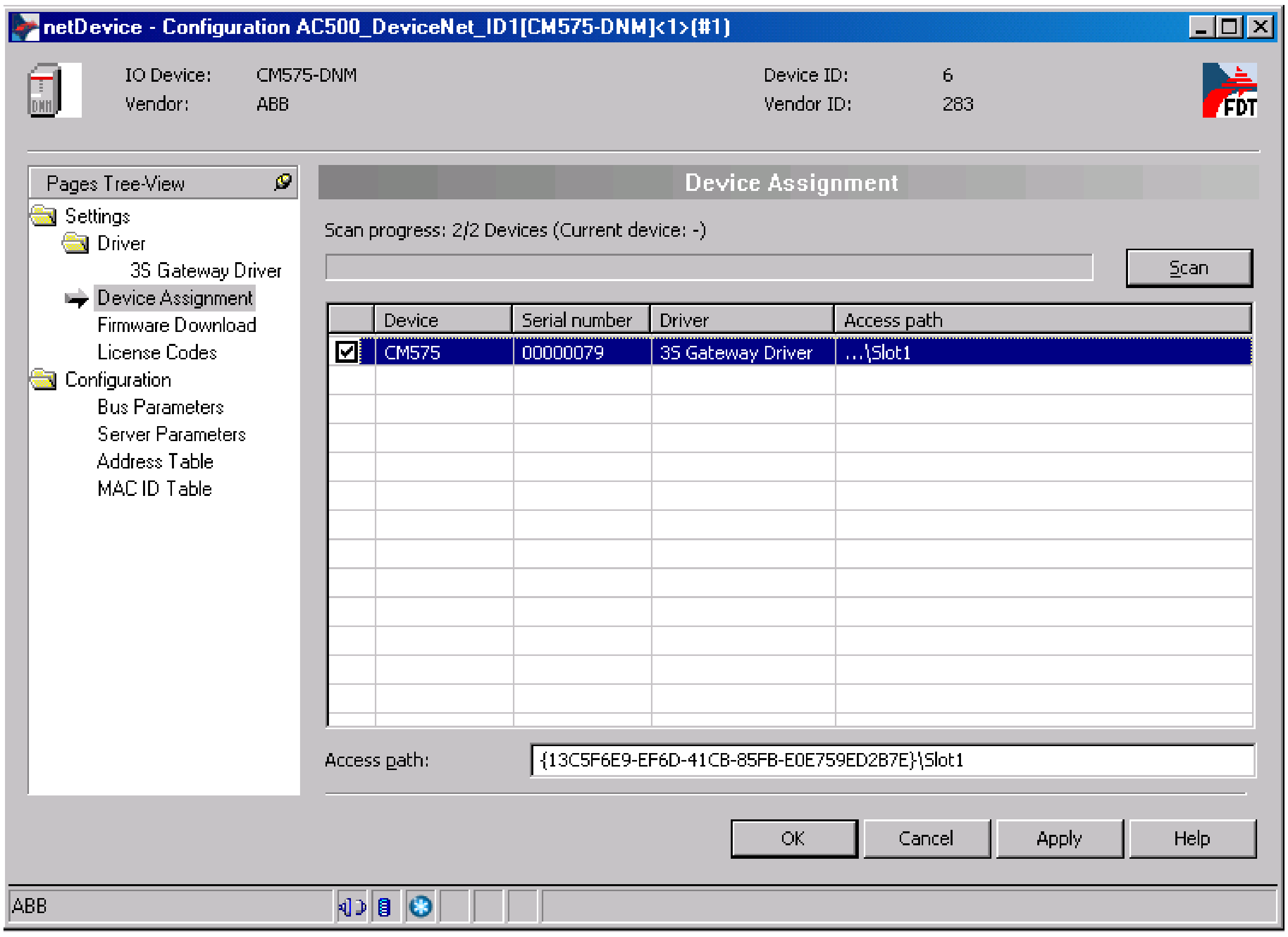

Kommunikationsparameter

Name	Wert	Kommentar
Port	COM4	
Baudrate	19200	
Parity	No	
Stop bits	1	
Motorola byteorder	Yes	

Serial (RS232)
AC500_AX522_Test.pro

Select the gateway and click on "OK".

Now the configuration tool SYCON.net searches for DeviceNet couplers that are connected to the selected interface. The tab "Device Assignment" in the DeviceNet master configuration shows the detected DeviceNet couplers.


netDevice - Configuration AC500_DeviceNet_ID1[CM575-DNM]<1>(#1)

IO Device: CM575-DNM
Vendor: ABB
Device ID: 6
Vendor ID: 283

Device Assignment

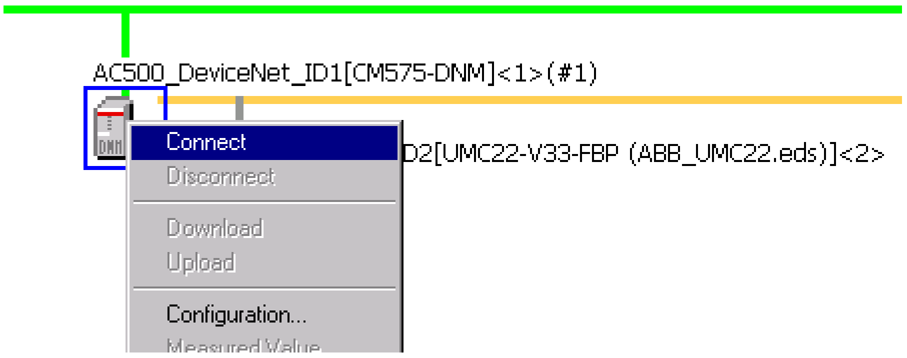
Scan progress: 2/2 Devices (Current device: -)

	Device	Serial number	Driver	Access path
☑	CM575	00000079	3S Gateway Driver	...\Slot1

Access path: {13C5F6E9-EF6D-41CB-85FB-E0E759ED2B7E}\Slot1

Select the desired coupler by marking the relevant checkbox and confirm your selection with "OK".

Select the "CM575" icon, press the right mouse button and select "Connect".

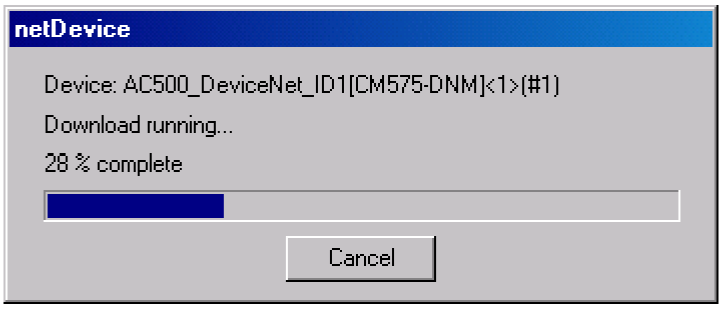

AC500_DeviceNet_ID1[CM575-DNM]<1>(#1)
D2[UMC22-V33-FBP (ABB_UMC22.eds)]<2>
Connect
Disconnect
Download
Upload
Configuration...
Measured Value

The CM575 icon is then highlighted by green background.

Set the controller to stop state.

Select the "CM575" icon, press the right mouse button and select "Download".

Confirm the appearing dialog with "Yes".


netDevice

Device: AC500_DeviceNet_ID1[CM575-DNM]<1>(#1)

Download running...

28 % complete

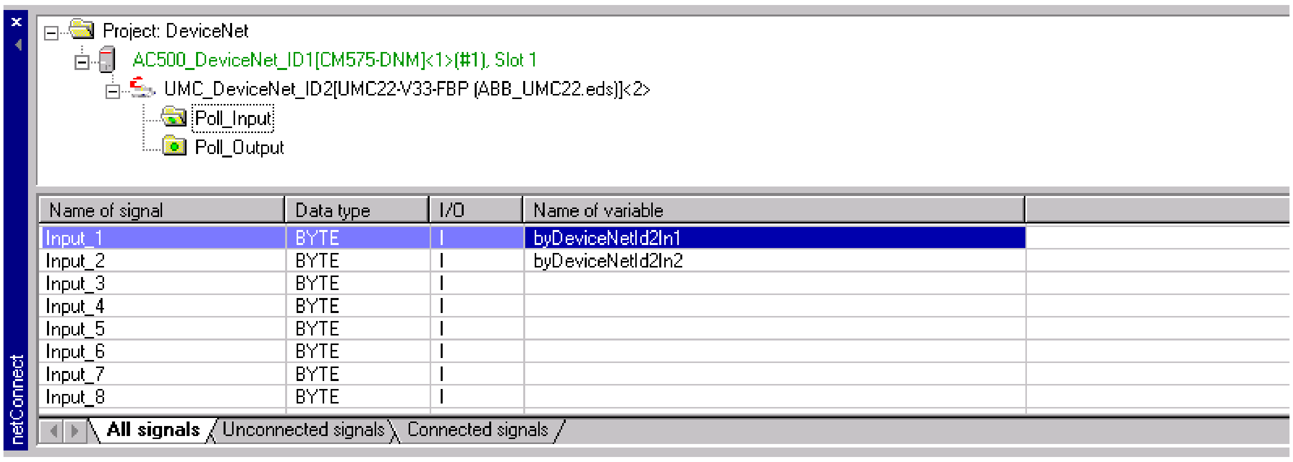
After successful completion of the download process, the "PWR" LED at the CM575 is on and the "RUN" LED flashes.

Select the "CM575" icon, press the right mouse button and select "Disconnect".

The coupler configuration is now completed.

To be able to use the DeviceNet data in the PLC program, you should assign corresponding variable names to the physical addresses using SYCON.net. These variables will then be available in the Control Builder and can be used directly there.

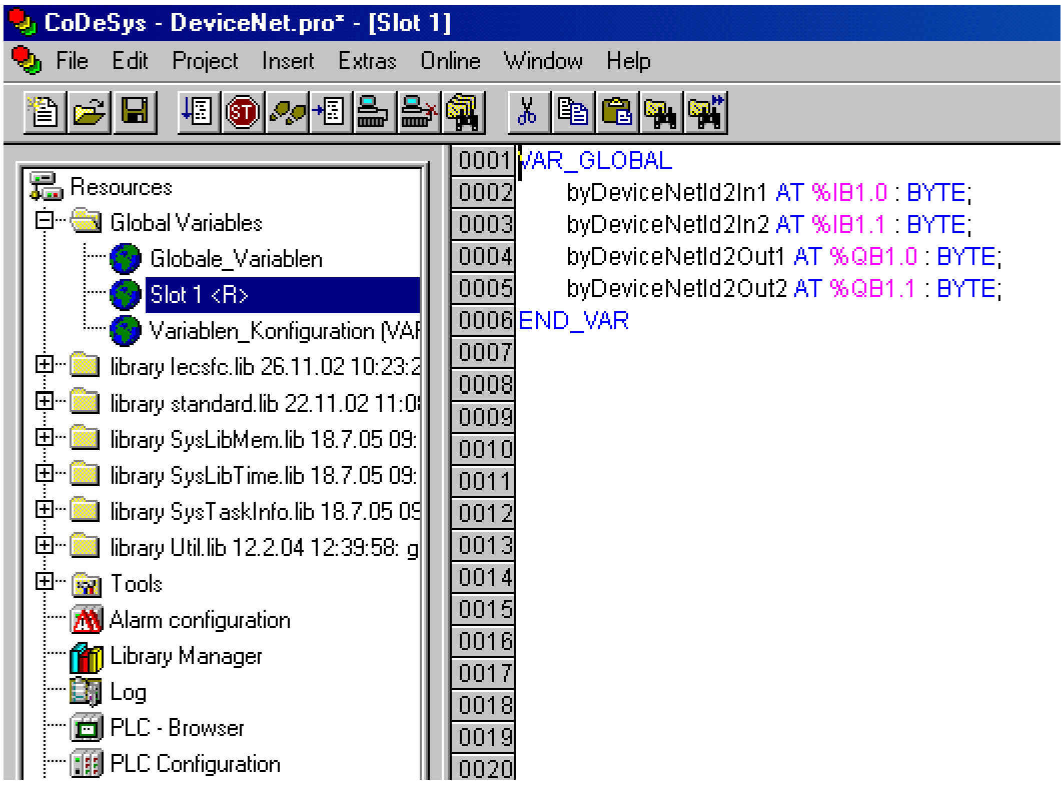
The assignment of the variable names is done in the netConnect window in SYCON.net.


Project: DeviceNet
AC500_DeviceNet_ID1[CM575-DNM]<1>(#1), Slot 1
UMC_DeviceNet_ID2[UMC22-V33-FBP (ABB_UMC22.eds)]<2>
Poll_Input
Poll_Output

Name of signal	Data type	I/O	Name of variable
Input_1	BYTE	I	byDeviceNetId2In1
Input_2	BYTE	I	byDeviceNetId2In2
Input_3	BYTE	I	
Input_4	BYTE	I	
Input_5	BYTE	I	
Input_6	BYTE	I	
Input_7	BYTE	I	
Input_8	BYTE	I	

The variable name has to be entered in the field "Variable Name". Double click to open the corresponding input field.

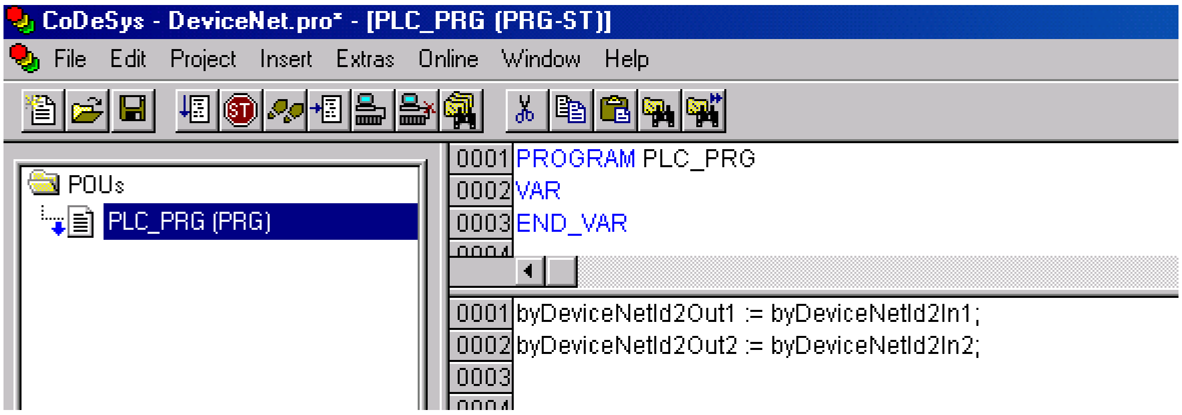
All variables declared here are automatically added to the "Global variables" folder when switching from SYCON.net to the Control Builder.


CoDeSys - DeviceNet.pro* - [Slot 1]
File Edit Project Insert Extras Online Window Help
Resources
Global Variables
Globale_Variablen
Slot 1 <R>
Variablen_Konfiguration (VAF
library Iecsfc.lib 26.11.02 10:23:2
library standard.lib 22.11.02 11:0
library SysLibMem.lib 18.7.05 09:
library SysLibTime.lib 18.7.05 09:
library SysTaskInfo.lib 18.7.05 09
library Util.lib 12.2.04 12:39:58: g
Tools
Alarm configuration
Library Manager
Log
PLC - Browser
PLC Configuration

```
VAR_GLOBAL
     byDeviceNetId2In1 AT %IB1.0 : BYTE;
     byDeviceNetId2In2 AT %IB1.1 : BYTE;
     byDeviceNetId2Out1 AT %QB1.0 : BYTE;
     byDeviceNetId2Out2 AT %QB1.1 : BYTE;
END_VAR
```

The declaration of the variables is now completed. The coupler variables can now be used in the user program.

The DeviceNet configuration is now tested by copying the DeviceNet slave ID2 inputs to the DeviceNet slave ID2 outputs in the user program


CoDeSys - DeviceNet.pro* - [PLC_PRG (PRG-ST)]
File Edit Project Insert Extras Online Window Help
POUs
PLC_PRG (PRG)

```
PROGRAM PLC_PRG
VAR
END_VAR
```

```
byDeviceNetId2Out1 := byDeviceNetId2In1;
byDeviceNetId2Out2 := byDeviceNetId2In2;
```

Now the user program can be downloaded to the controller and started.

During exchange of DeviceNet data, the "RUN" LED at the CM575 continuously lights up.

1.5.2 Running operation

The DeviceNet protocol is automatically processed by the coupler and the operating system of the controller. The coupler is only active on the bus if it has been initialized correctly before and if the user program is running. No function blocks are necessary for exchanging process data via DeviceNet. Special DeviceNet functions can be realized using the function blocks of the DeviceNet library.

The coupler starts communication via DeviceNet after the user program is started and then attempts to initialize the configured slaves. After a successful initialization, the slave exchanges the process data. The exchange of I/O data with the slaves is done automatically.

If the user program is stopped, the coupler shuts down the DeviceNet communication in a controlled manner.

1.5.3 Error diagnosis

DeviceNet communication errors are indicated by the coupler LEDs. Malfunctions of the DeviceNet driver or the coupler itself are indicated by the corresponding error class in the PLC (refer to System Technology of the CPU / The diagnosis system in AC500). Furthermore, the DeviceNet library provides different function blocks which allow detailed error diagnosis (refer to "The DeviceNet Library").

1.5.4 Function blocks

Libraries:

DeviceNet_AC500_V11.lib

Group: Status / Diagnosis	
DNM_DEV_DIAG	Polling diagnosis data from a slave
DNM_STATE	Reading the DeviceNet coupler status
DNM_SYS_DIAG	Displaying status surveys of all slaves

Group: Parameters	
DNM_GET_ATTR	Reading an attribute from a slave object
DNM_RES_OBJ	Resetting a slave object
DNM_SET_ATTR	Writing an attribute to a slave object

1.6 Diagnosis

1.6.1 Status LEDs

LED	Color	Status	Meaning
PWR	green	ON	Voltage on
		OFF	Voltage off
RDY	yellow	ON	Coupler ready
		flashes cyclic	Bootstrap loader active
		flashes non-cyclic	Hardware or system error
		OFF	Defective hardware or voltage off
RUN	green	ON	Communication in progress
		flashes cyclic	Communication stopped
		flashes non-cyclic	Missing or faulty configuration
		OFF	No communication or voltage off
NET/ MOD	green/ red	ON (green)	Device is in online mode and has established one or several connections
		flashes cyclic (green)	Device is in online mode and has established no connection
		flashes cyclic (green/red)	Faulty communication
		ON (red)	Critical connection error, device has detected a network error (duplicated MAC ID or bus interrupted)
		flashes cyclic (red)	Connection timeout
		OFF	After start of the device and during duplicate MAC ID check

Table: Status LEDs of the DeviceNet coupler

1.6.2 DeviceNet error messages

The DeviceNet error messages are listed in the section 'Error messages for the block libraries'.

1.6.3 Function blocks

DeviceNet master

Refer to 1.5.4 Function blocks

Online diagnosis

Refer to the documentation for the field bus configuration tool SYCON.net

1.7 Further information

1.7.1 Standardization

BOSCH CAN Specification - Version 2.0, Part A

ISO 11898

ODVA DeviceNet Specification Release 2.0, Errata 1 & 2

1.7.2 Important addresses

Open DeviceNet Vendor Association (ODVA)
PMB 499
20423 State Road 7 #F6
Boca Raton, FL 33498-6797
U.S.A.

Phone: (+1) 954 340-5412
Fax: (+1) 954 340-5413
Internet: http://www.odva.org

DeviceNet Europe
c/o Teja Ulrich
Elektrastraße 14
D-81925 München
Germany

Phone: (+49) 8991049571
Fax: (+49) 8991049573
E-Mail: Teja.Ulrich@munich.netsurf.de

1.7.3 Terms, definitions and abbreviations

ACK	Acknowledged Exchange
ODVA	Open DeviceNet Vendor Association
CAN	Controller Area Network
COS	Change Of State
EDS	Electronic Data Sheet
MAC ID	Media Access Control Identifier, bus address
Trunk Line	Main strand of the bus
Drop Line	Branch line
UCMM	Unconnected Message Manager

2 Index

Manual No.: 2CDC 125 022 M0201

ABB STOTZ-KONTAKT GmbH
Eppelheimer Straße 82   69123 Heidelberg, Germany
Postfach 10 16 80   69006 Heidelberg, Germany

Telephone	(06221) 701-0
Telefax	(06221) 701-240
Internet	http://www.abb.de/stotz-kontakt
E-Mail	desst.helpline@de.abb.com