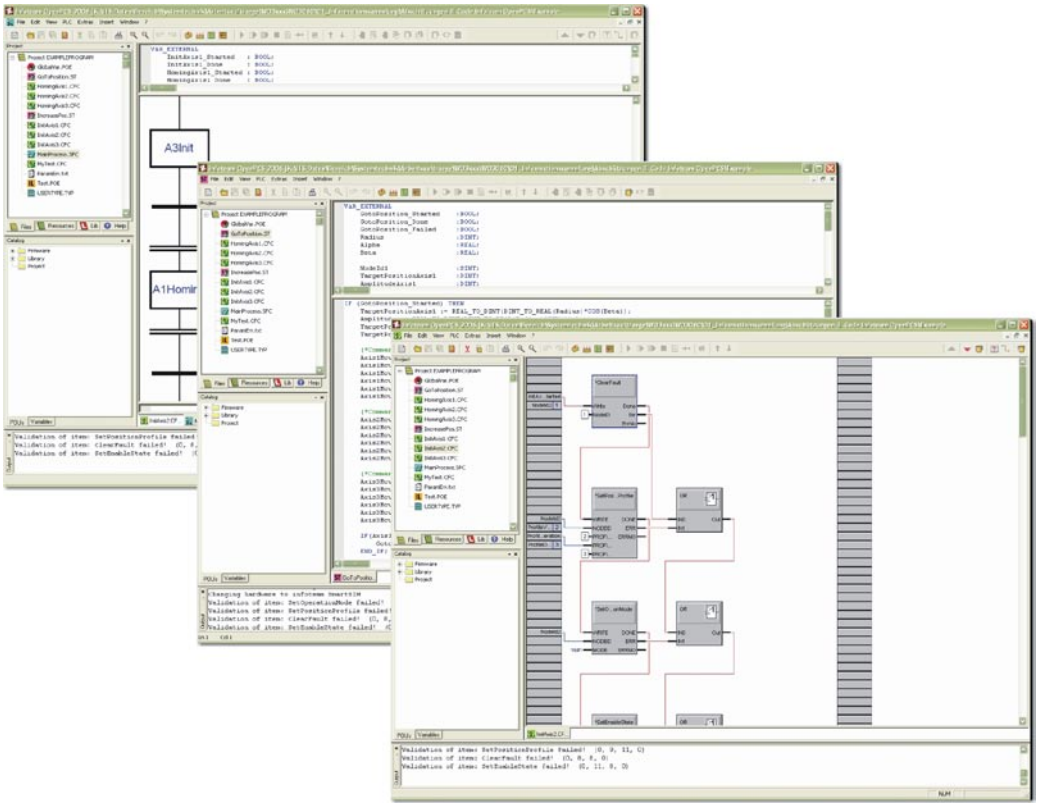
EPOS P
MCD EPOS P

Programmable Positioning Controller

Documentation

Programming Reference



maxon document 786912-06

1 Table of Contents

2 Table of Figures

3 Introduction

This documentation 'Programming Reference' provides the information about programming the EPOS P 24/5 and the MCD EPOS P 60 W positioning controller. On the basis of an example it describes the standard procedure of writing and debugging an IEC-61131 program. Additionally the motion control function blocks are described.

The latest edition of this "Programming Reference", additional documentation and software to the EPOS P positioning controller may also be found on the internet in www.maxonmotor.com category <Service & Downloads>.

4 How to use this guide

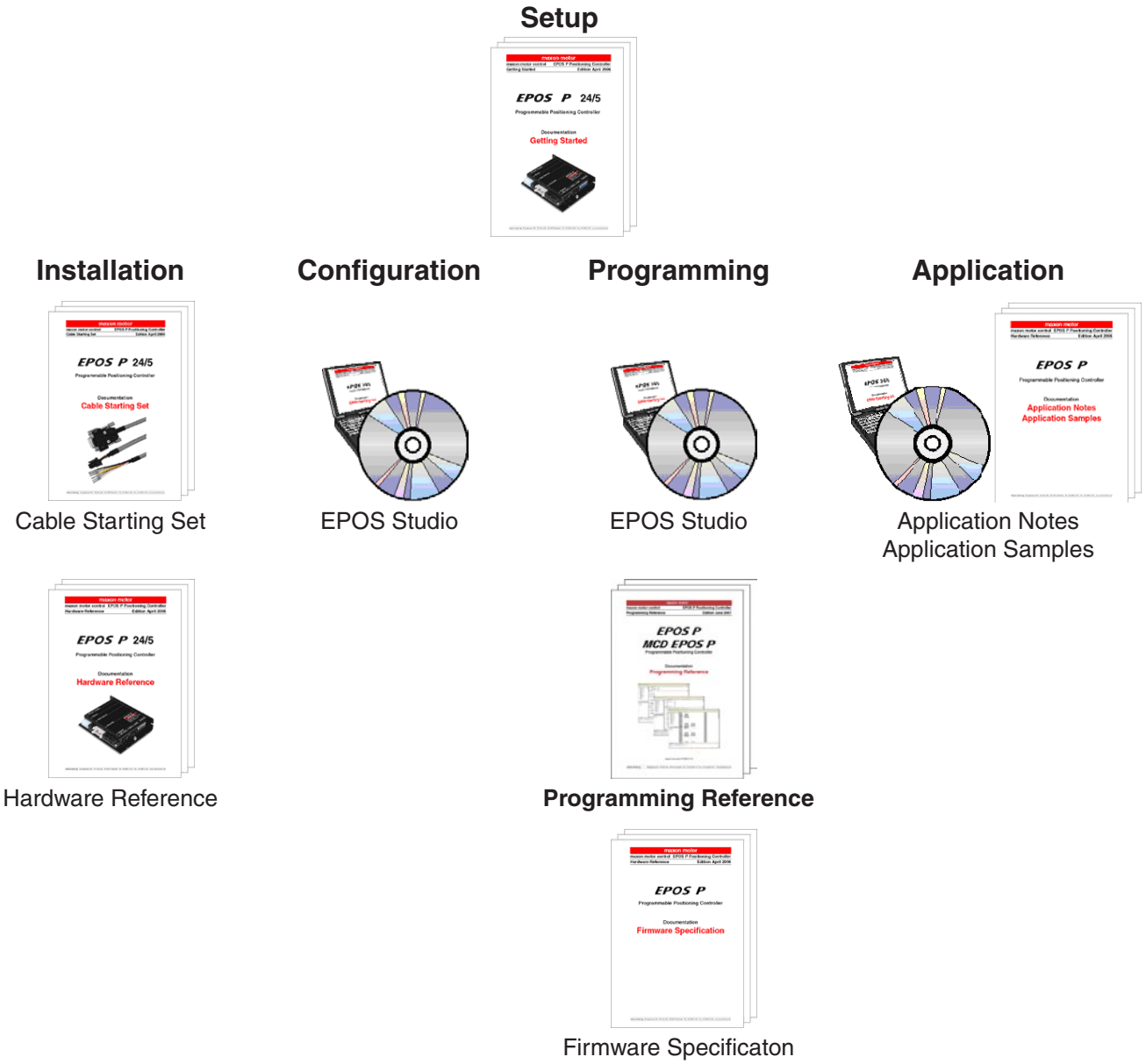


Figure 1: EPOS P documentation hierarchy

5 Start Programming

5.1 Open Programming Tool

Use the 'EPOS Studio' and make sure a project (*.pjm), containing an EPOS P or an MCD EPOS P device, is loaded. This devices allows you to open the programming tool.

a) Click the page 'Tools' in the page navigator window.

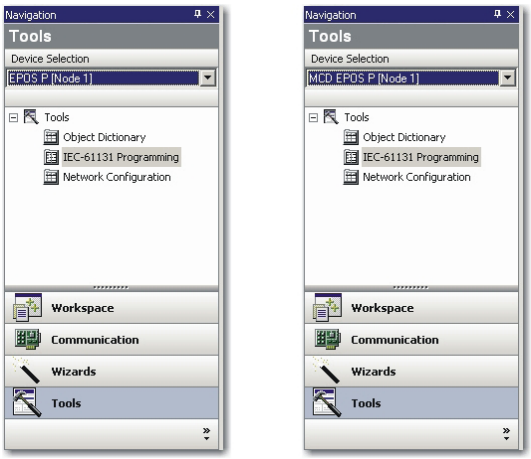


Figure 2: Page Navigator Window

b) Select the EPOS P or MCD EPOS P device in the device selection combo box.

c) Double click the tool 'IEC-61131 Programming'. A view containing a list of sample projects is opening. This view can be used as a control centre to open projects and to control the program state.

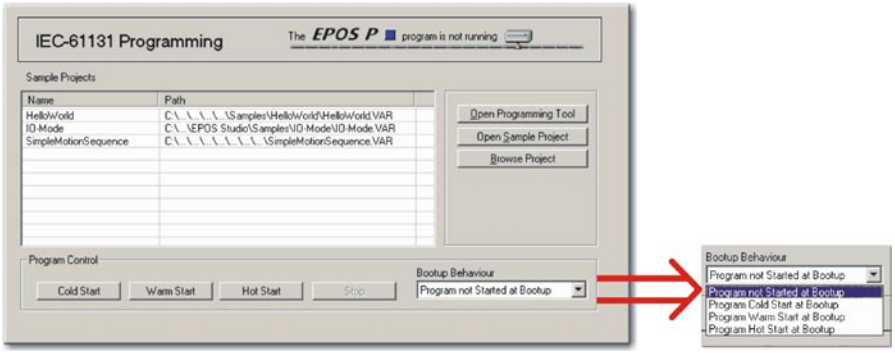


Figure 3: 'IEC-61131 Programming' Window

Project

- Open Programming Tool ..... External Tool 'OpenPCS' is opened
- Open Sample Project ..... Selected project is opened
- Browse Project ..... Search for a IEC-61131 project (*.VAR) to open

Program Control

- Cold Start ..... Start the program initializing the variables to their default values
- Warm Start ..... Restart the program at the beginning and restore the values
- Hot Start ..... Restart the program at the place where the program was stopped and restore the values
- Stop ..... Stop the program

Bootup Behaviour
The bootup behaviour of a downloaded program after a reset can be pre-selected.

d) Click the button 'Open Programming Tool' to open the external tool 'OpenPCS'.

5.2 Licence Key Configuration

In order to be able to use the OpenPCS programming tool a valid licence key has to be configured. Open the Licence Dialog to check if a valid licence is available.

a) Open the menu ‘Extras’ and the submenu ‘Tools’ to select the menu Item ‘Licence’.

b) If already a valid license is registered, skip the next step.

c) If no license is registered enter a valid serial number and a licence code. The license code is delivered together with the EPOS Studio in the ‘ReadMe.txt’ file.

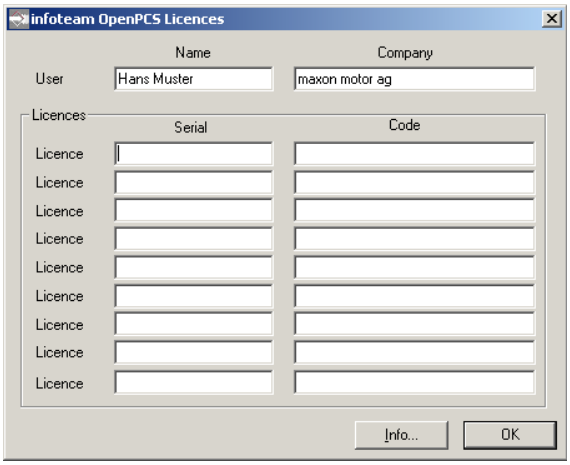


Figure 4: License Dialog

Remark

If the licence code is out of date, download the latest EPOS Studio from the internet: www.maxonmotor.com, category <Service & Downloads>

5.3 Connection Setup

Configure a connection to establish an online connection to the EPOS P device. This connection configuration is used for downloading, debugging and controlling the program.

a) Open the menu ‘PLC’ and select the menu item ‘Connections…’

b) If a connection entry ‘ProxyEpos’ is available click the button ‘Edit’ and continue with step ‘f’.

c) If no connection entry ‘ProxyEpos’ is available click the button ‘New’ to add a new connection.

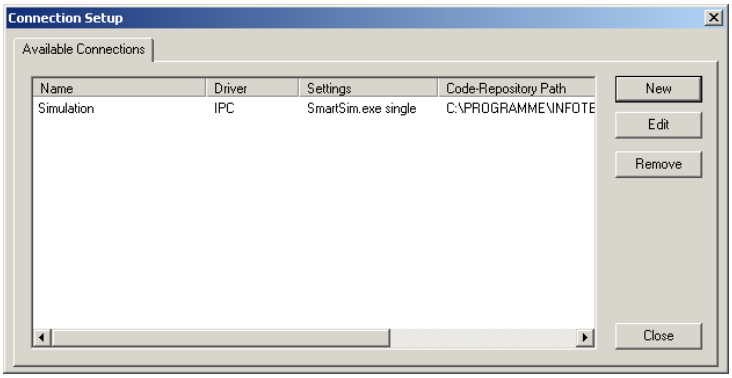


Figure 5: Connection Setup

d) Enter the connection name ‘ProxyEpos’ and add comment line.

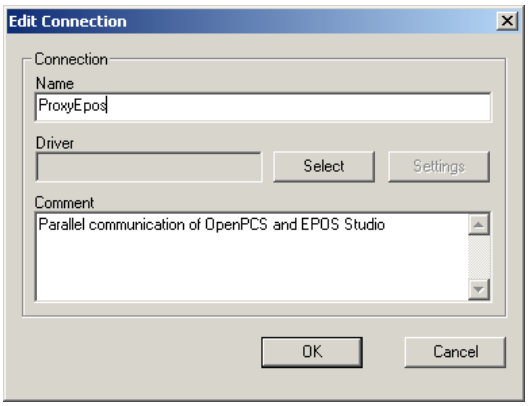


Figure 6: Edit Connection

e) Press the ‘Select’ button to choose a communication driver. Select the driver ‘ProxyEpos’. This driver allows a parallel communication of the EPOS Studio and the programming tool ‘OpenPCS.

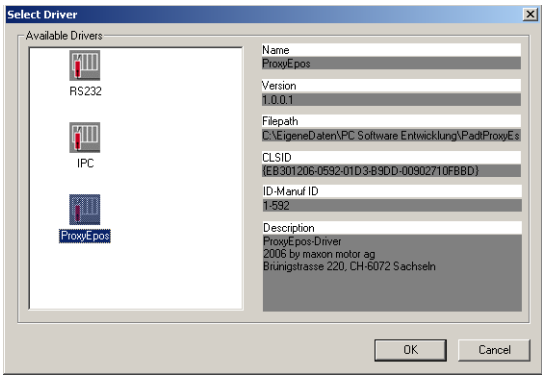


Figure 7: Select Driver

f) Click the button ‘Settings’ to select the correct RS-232 port and baudrate. The default baudrate is 115200 bps.

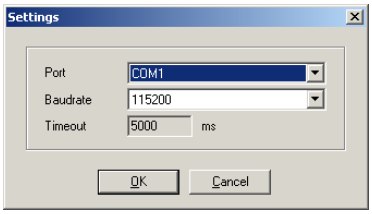


Figure 8: Settings

g) Close the dialogue ‘Settings’ and the dialogue ‘Edit Connection’. The following connection entry should be available now.

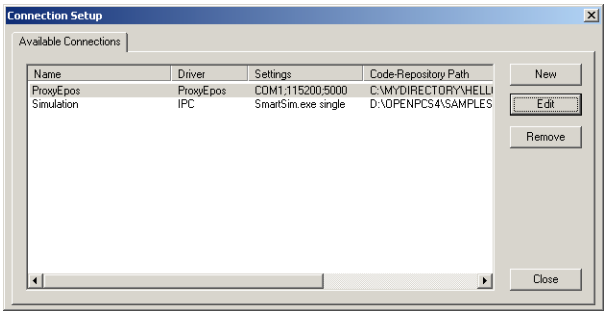


Figure 9: Connection Entry ‘ProxyEpos’

5.4 Sample Project: HelloWorld

The following chapters explain the standard procedure writing a program. This procedure is documented considering an example.

This example is a very simple program without any motion control features. The intention of this program is only to show the handling of the programming tool. The program is counting up and downward. Reaching the maximum value the text ‘HelloWorld’ is written to the variable ‘Text’.

For an example using motion control functionality have a look at the chapter Example Projects.

```
PROGRAM Counter
VAR
       UpCounting     : BOOL := TRUE;
       Count          : UINT := 0;
       CountMax       : UINT := 300;
       Text           : STRING;
END_VAR

(*Update UpCounting*)
IF (Count = 0) THEN
       UpCounting := TRUE;
       Text := ' ';
END_IF;

IF (Count >= CountMax) THEN
       UpCounting := FALSE;
       Text := 'HelloWord';
END_IF;

(*Do Counting*)
IF (UpCounting) THEN
       Count := Count + 1;
ELSE
       Count := Count -1;
END_IF;

END_PROGRAM
```

5.5 Creating New Project

a) Open the menu ‘File’ and the submenu ‘Projects’ and select the menu item ‘New’.

b) Select the file type ‘maxon motor ag’ and the template ‘EPOS P Project’ or ‘MCD EPOS P Project’.

c) Enter the project name ‘HelloWorld’ and browse the location to store the new project.

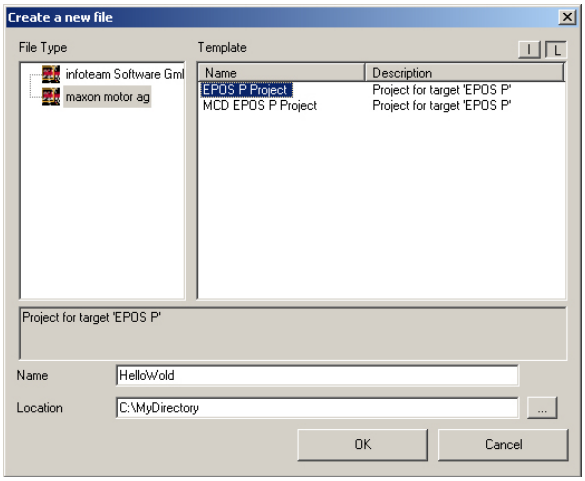


Figure 10: Create New Project

d) Click okay and a new project is created. This project contains a resource item containing a configuration for the hardware module ‘maxon motor EPOS P’ and a network connection ‘ProxyEpos’. The resource specification can be checked by clicking the menu item ‘Resource Properties’ in the menu ‘PLC’.

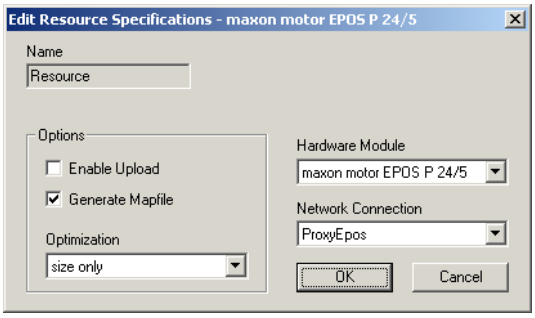


Figure 11: Edit Resource Specifications

5.6 Writing Program Code

a) As a first step you have to add a new program to the project. Click the menu ‘File’ and the menu item ‘New’ to open the dialogue ‘Create a new file’.

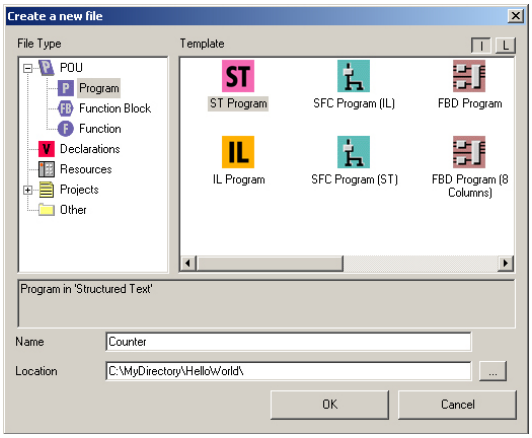


Figure 12: Create Program File

Select the file type ‘Program’ in the directory POU (Program Organisation Unit). Choose the preferred programming language for your program. Select for this example the programming language ‘Structured Text’. Enter the name ‘Counter’ and close the dialogue with ‘OK’.

b) The message dialogue is opened and you are asked whether you want to add the program item ‘Counter’ to the active resource or not. Answer this question with ‘Yes’.

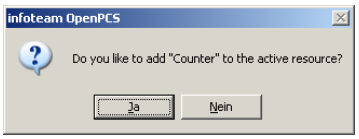


Figure 13: Add to active resource

c) Do the configuration of the program ‘Counter’. Open the tab ‘Resources’, select the task item ‘Counter’ and open the properties via the context menu. Select the task type ‘Timer’ and set the Time to 10 ms.

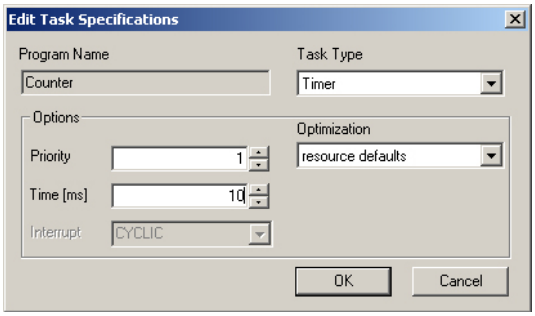


Figure 14: Task Specifications

d) Now you are ready to start programming. Open the program item ‘Counter.ST’.

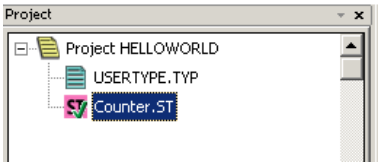


Figure 15: Project HelloWorld

e) Enter the variable declaration

```
VAR_EXTERNAL

END_VAR

VAR_GLOBAL

END_VAR

VAR
    UpCounting  : BOOL := TRUE;
    Count       : UINT := 0;
    CountMax    : UINT := 300;
    Text        : STRING;
END_VAR
```

Figure 16: Variable Declaration

f) Enter the program code

```
(*Update UpCounting*)
IF (Count = 0) THEN
    UpCounting := TRUE;
    Text := '';
END_IF;
IF (Count >= CountMax) THEN
    UpCounting := FALSE;
    Text := 'HelloWorld';
END_IF;

(*Do Counting*)
IF (UpCounting) THEN
    Count := Count + 1;
ELSE
    Count := Count - 1;
END_IF;
```

ST Counter.ST ...

Figure 17: Program Code Implementation

g) To verify the correctness of the implementation execute an syntax check. Click the menu ‘File’ and select the menu item ‘Check Syntax’.

5.7 Compiling and Executing Program Code

a) After code implementation the program has to be compiled. Click the menu ‘PLC’ and select the menu item ‘Build Active Resource’. The following logging output should appear in the output window.

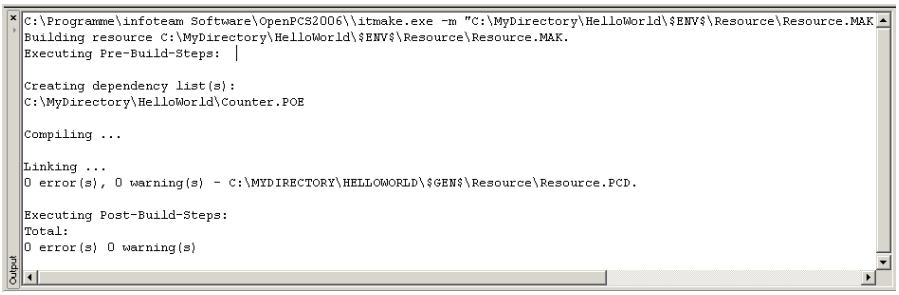


Figure 18: Output Window

b) In order to download the program code, an online connection has to be established. Click the menu ‘PLC’ and select the menu item ‘Online’. If a new code is detected, you will be asked if you want to download the current resource. Click ‘Yes’ in order to update the program on the EPOS P.

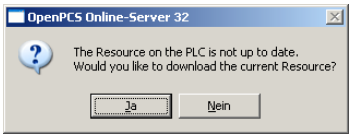


Figure 19: Download new code

c) Start the downloaded code by clicking the menu ‘PLC’ and selecting the menu item ‘Cold Start’.

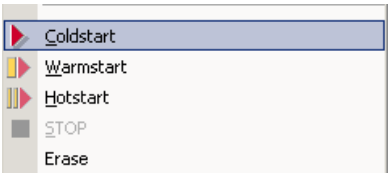


Figure 20: Cold Start

5.8 Debugging Program Code

a) Add a watch variable to the window ‘Test And Commissioning’. Open the tab ‘Resources’ in the project window. Open the tree view of the task ‘COUNTER’ and select the variable ‘COUNT’. Select the command ‘Add To Watchlist’ in the context menu. The variable ‘COUNT’ is added to the window ‘Test And Commissioning’.

b) Add also the variables ‘UPCOUNTING’ and ‘COUNTMAX’

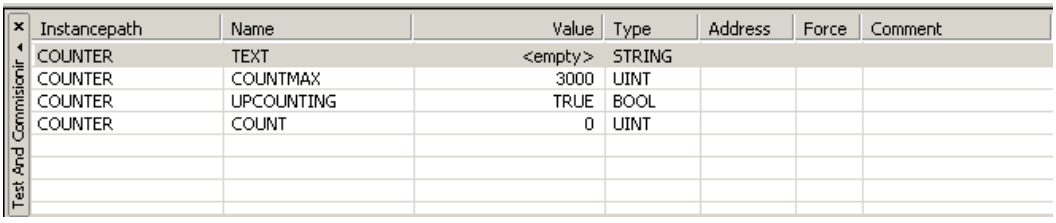


Figure 21: ‘Test And Commissioning’ Window

c) For a step by step program debugging add a breakpoint to the program code. Put the mouse cursor to the line where you want to add the breakpoint. Click the menu ‘PLC’ and the submenu ‘Breakpoint’ and select the menu item ‘Toggle’. The program is stopping at the breakpoint.

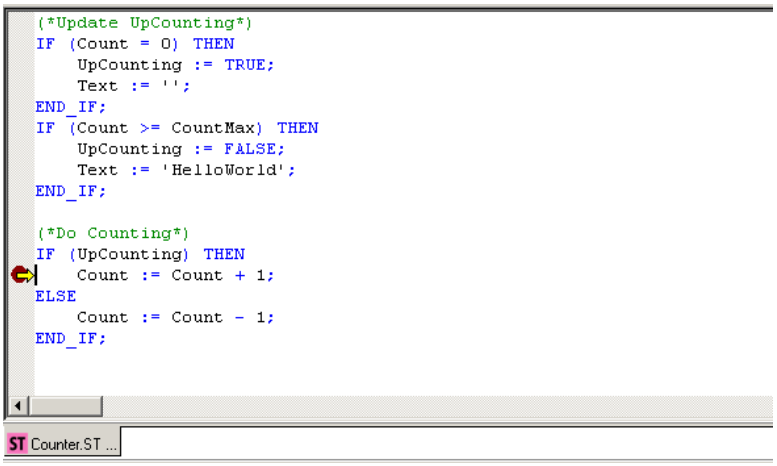


Figure 22: ‘Breakpoint’

d) Delete the breakpoint by toggling the breakpoint. Continue the program execution by clicking the menu ‘PLC’ and the submenu ‘Breakpoint’ and select the menu item ‘Go’.

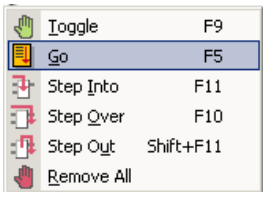


Figure 23: Continue Program Execution

6 Project Settings

The following chapters explain the function of some project specific settings which have to be done during the programming process.

6.1 Resource Properties

In general, a resource is equivalent to a PLC or a micro controller. A resource definition consists of a name for identification, the hardware description, i.e. information about the properties of your PLC which will be used by OpenPCS, and a connection name, i.e. information about the kind of communication between OpenPCS and the control system.
A resource maintains a list of tasks which are to be run on the control system.

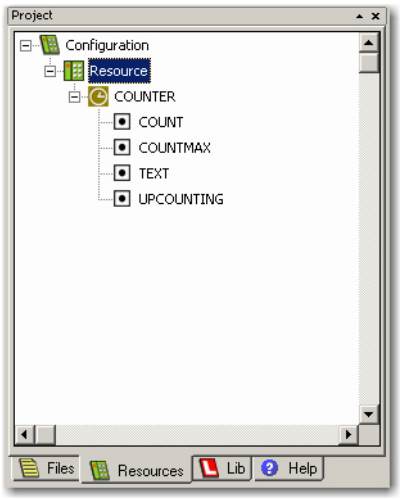


Figure 24: Resource-Pane

6.1.1 Edit Resource Properties

To edit a resource, right-click on it and choose ‘Properties’ in the context-menu.
A dialog-box will open, where you can change the following properties:

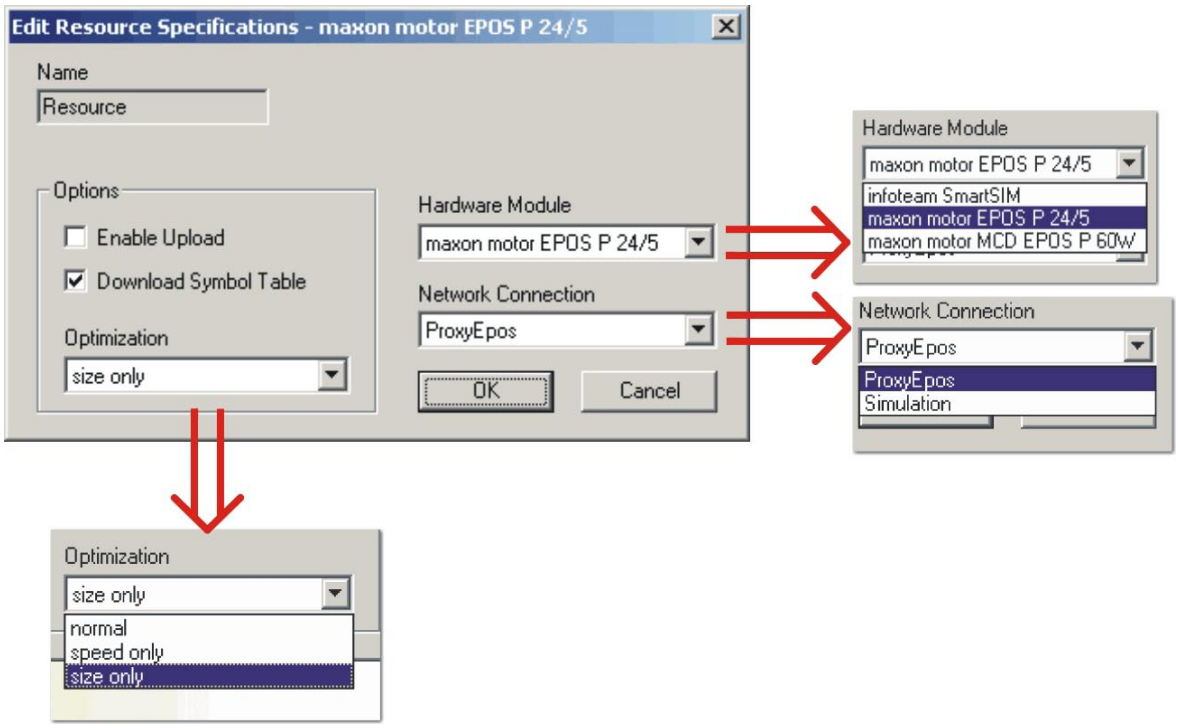


Figure 25: Resource Specifications window

Hardware Module:

Under 'Hardware Module', select the configuration file corresponding to the controller you are using. In case of using maxon hardware, the modules 'maxon motor EPOS P 24/5' and 'maxon motor MCD EPOS P 60 W' are available. To use the windows simulation SmartSIM, use 'SmartSIM'.

Network Connection:

Under 'Network Connection', select the communication connection to connect to your resource. To communicate with maxon EPOS P 24/5 or MCD EPOS P 60 W choose 'ProxyEpos'. To work with the PLC-Simulation of OpenPCS select 'Simulation'.

Options:

- Enable Upload:
  Not supported.

- Download Symbol Table:
  No effect.

Optimization:

OpenPCS supports optimization settings 'speed', 'size' and 'normal'.

- size only:
  Compiler option, that optimizes the generated code on code size.

- speed only:
  Compiler option, that optimizes the generated code on speed.

- normal:
  Mix between size only and speed only.

Remark:
Take in mind that full debugging is only possible with Optimization option size only!

6.2 Task Properties

In general, a task is equivalent to a program. The definition of a task consists of the name, the information about the execution of the task and a POU of type PROGRAM which should be executed in this task.

6.2.1 Edit Task Properties

To edit a task, right-click on it and choose 'Properties' in the context-menu.
A dialog-box will open, where you can change the following properties:

Task Type:

OpenPCS supports all three tasks types defined by IEC-61131-3.

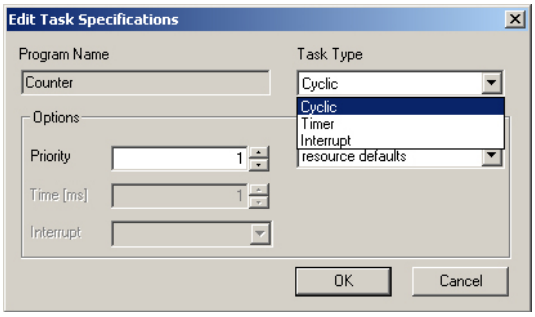


Figure 26: Edit Task Specification - Task Type

- Cyclic:
  Cyclic tasks will be executed when no timer or interrupt tasks are ready to run. The priority that can be specified in the task properties will be interpreted as a cycle interleave, e.g. priority = 3 will have this task executed only every third cycle. No particular execution order is defined by OpenPCS amongst multiple cyclic tasks.

- Timer:
  Timer tasks will be executed every n milliseconds, with n specified in the task properties.

- Interrupt:
  Interrupt tasks will be executed as soon as the interrupt occurs they are linked to.

Optimization:

OpenPCS supports optimization settings „speed", „size" and „normal".

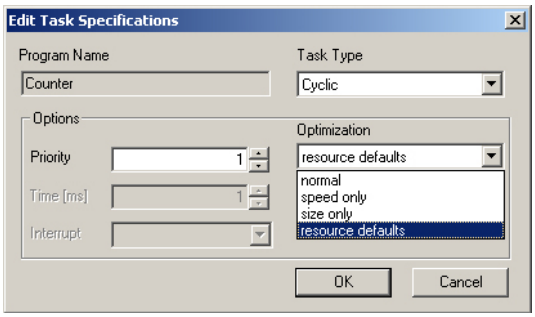


Figure 27: Edit Task Specification - Optimization

- resource default:
  This task uses the Optimization attributes of the resource.

- size only:
  Compiler option, that optimizes the generated code on code size.

- speed only:
  Compiler option, that optimizes the generated code on speed.

- normal:
  Mix between size only and speed only.

Remark:
Take in mind that full debugging is only possible with Optimization option size only!

Interrupt

This task type is only executed at particular interrupt events. The type of the event is selected with the option Interrupt.

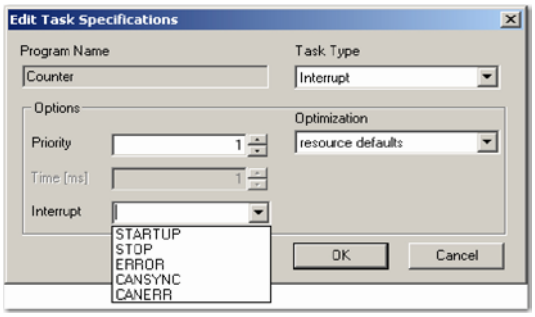


Figure 28: Edit Task Specification - Interrupt

- STARTUP:
  The task with type Interrupt and options startup is executed once at startup
  Remark: Function Blocks needs typically more than one cycle to finish

- STOP:
  The task with type Interrupt and option Stop is executed once at program stop
  Remark: Function Blocks needs typically more than one cycle to finish

- ERROR:
  The task with type Interrupt and option Error is executed once at program error
  Remark: Function Blocks needs typically more than one cycle to finish

- CANSYNC:
  The task with type Interrupt and option CanSync is synchronized with CANopen SYNC
  Remark: The interrupt source for this task is the CANopen SYNC Cycle, the task will never be called when the SYNC Master is not activated

- CANERR:
  The task with type Interrupt and option CanError is synchronized with CANopen EMCY
  Remark: The interrupt source for this task is the CANopen EMCY, this task is called once when a connected CANopen Slave reports a Error with CANopen EMCY

6.3 Network Configuration

The 'Network Configuration' tool is used to set up a multi axis network. Use this tool to configure the master and slave devices to be used in a multi axis IEC-61131 program.

6.3.1 Overview

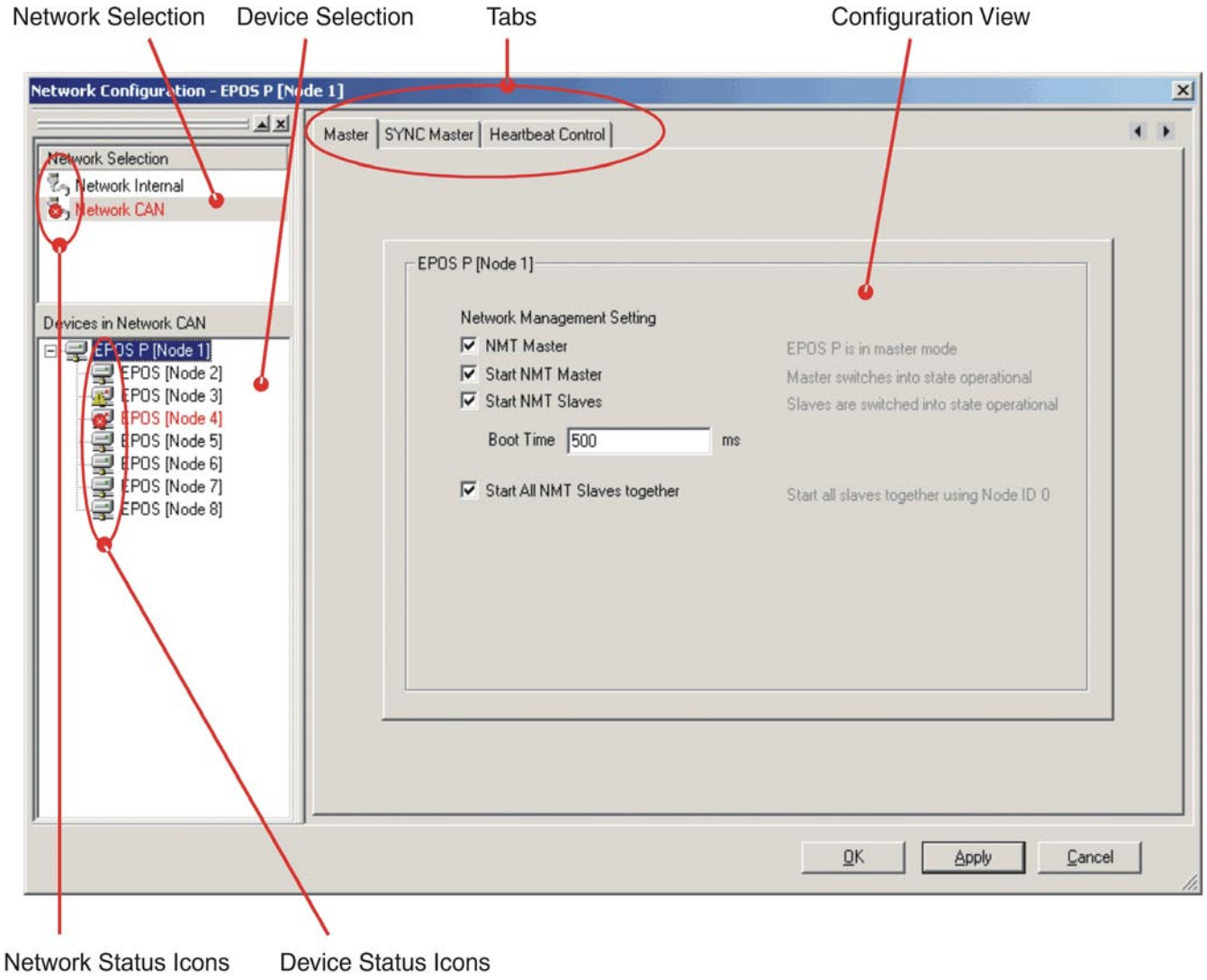


Figure 29: Network Configuration Overview

Network Selection	Displays all available networks		
Device Selection	Display all available devices in the network selected		
Tabs	Selection of configuration views		
Configuration View	Configuration view to change settings		
Network Status Icons		Okay	There's no error or warning in this network
		Warning	There are warnings in this network. Check the devices
		Error	There are errors in this network. Check the devices
Device Status Icons		Okay	There's no error or warning in this device configuration
		Warning	There are warnings in this device configuration. Check the configuration views
		Error	There are errors in this device configuration. Check the configuration views

6.3.2 Master Configuration

For the master configuration select the master item in the device selection. The master has to be configured for all networks.

6.3.2.1 Configuration View „Master“

The configuration view “Master” allows to define the behaviour of the master device.

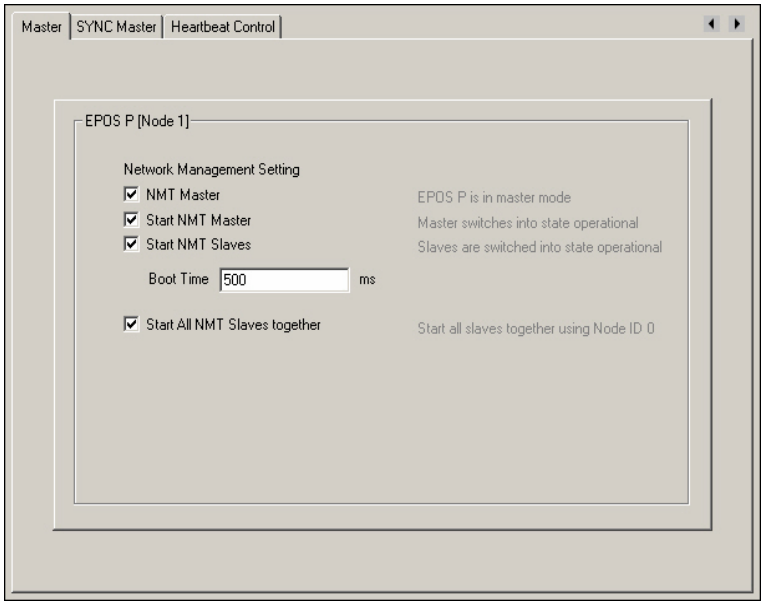


Figure 30: Configuration View ‘Master’

Option	Default	Description
NMT Master	Checked	EPOS P is in master mode and is able to communicate with slaves
Start NMT Master	Checked	After bootup, the master is switching into NMT state operational
Start NMT Slaves	Checked	After bootup, the master is switching the slaves into NMT state operational
Boot Time	500 ms	Time to wait before addressing slaves after reset
Start All NMT Slaves together	Checked	All slaves are starting together using a broadcast service

6.3.2.2 Configuration View „SYNC Master“

The configuration view “SYNC Master” allows to define the behaviour of the SYNC Master in the network. The SYNC Master must be active if any synchronous PDO is configured.

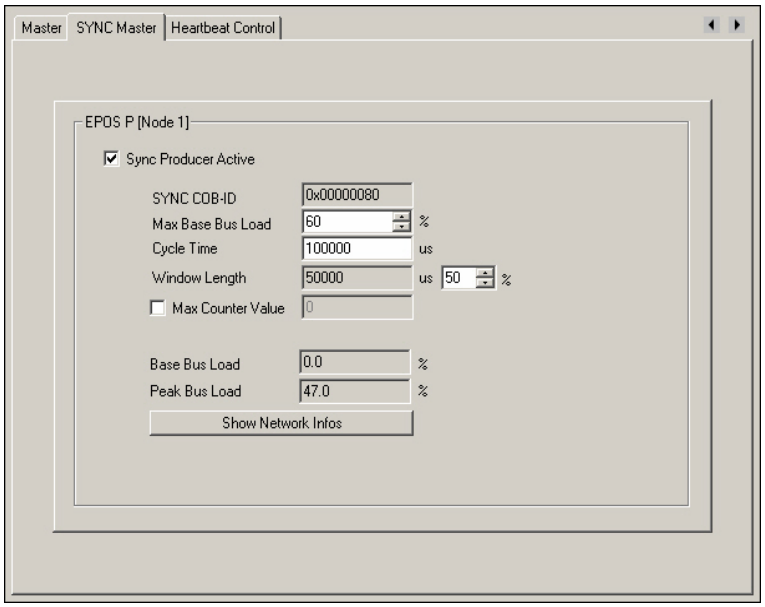


Figure 31: Configuration View ‘SYNC Master’

Option	Default	Description
Sync Producer Active	Checked	Enable or Disable the SYNC Master
SYNC COB-ID	0x00000080	COB-ID of the SYNC CAN Frame
Max Base Bus Load	60%	Recommended Maximum Base Bus Load
Cycle Time	100’000 us	Cycle Time of the SYNC CAN Frame
Window Length	50%	Window for sending and receiving synchronous PDOs
Max Counter Value	Disabled	Enable or disable sending a SYNC CAN Frame including a data byte containing a counter value

Calculations	Description
Base Bus Load	Calculated bus load containing CAN frames, that are cyclically transmitted. The following CAN frames are included in calculation: SYNC, PDO sync, Heartbeat
Peak Bus Load	Calculated bus load containing all CAN frames, that are transmitted. The following CAN frames are included: SYNC, PDO sync, Heartbeat, PDO async **Remark:** The asynchronous PDO’s are a potential risk for a bus overload. Use the ‘Inhibit Time’ to limit the transmission rate.

How To Reduce Bus Load

In case of a bus load higher than the maximum bus load, the transmission of CAN frames has to be limited. Use one of the following action to reduce the bus load.

Action	Objects	Description / Effect	
Increase CAN Bitrate	All Objects	The CAN Bitrate can be increased up to 1Mbit/s. Consider the maximum allowed Bitrate for your network length.	
		Bitrate	**Max. line length according to CiA DS-102**
		1 Mbit/s	25 m
		800 kBit/s	50 m
		500 kBit/s	100 m
		250 kBit/s	250 m
		125 kBit/s	500 m
		50 kBit/s	1000 m
		20 kBit/s	2500 m
Increase Cycle Time	SYNC, PDO sync	The cycle time of the SYNC producer may be increased to reduce the bus load. Increasing the cycle time is reducing the update rate of network variables in your IEC-61131 program.	
Increase Heartbeat Producer Time	Heartbeat	Increase the producer time of the heartbeat CAN frames. Increasing the producer time is reducing the reaction time to a broken CAN bus.	
Increase Inhibit Time	PDO async	Increase the inhibit time of the asynchronous PDO's. Increasing the inhibit time is reducing the update rate of network variables in your IEC-61131 program.	

For more details click the button 'Show Network Infos':

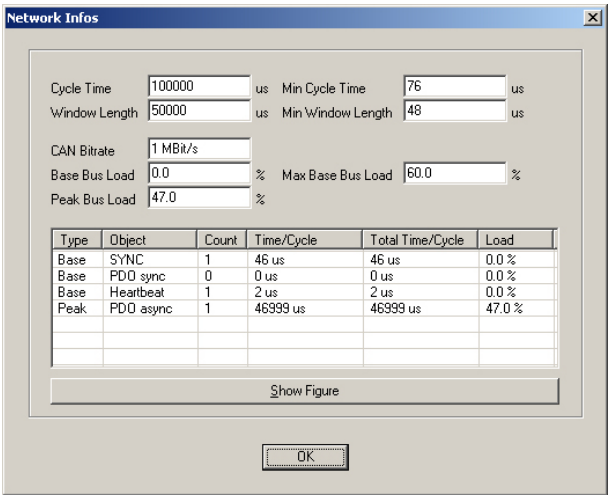


Figure 32: Network Infos

Info	Description
Cycle Time	Configured Cycle Time
Min Cycle Time	Min Cycle Time calculated based on the maximum base bus load.
Window Length	Configured Window Length
Min Window Length	Min Window Length calculated based on the maximum base bus load.
CAN Bitrate	Configured CAN Bitrate

Info	Description
Base Bus Load	Calculated bus load containing CAN frames, that are cyclically transmitted. Have a look at the detailed load table to see what types of CAN frames are included in calculation.
Max Base Bus Load	Recommended Maximum Base Bus Load
Peak Bus Load	Calculated bus load containing all CAN frames, that are transmitted. Have a look at the detailed load table to see what type of CAN frame is included in calculation. **Remark:** The asynchronous PDO's are a potential risk for a bus overload. Use the 'Inhibit Time' to limit the transmission rate.

Table Column	Description
Type	Base: Bus load of this object is added to the base and peak bus load. Peak: Bus load of this object is added only to the peak bus load.
Object	Type of CAN frame transmitted
Count	Number of CAN frames transmitted
Time/Cycle	Time to transmit one CAN frame per cycle time. **Remark:** For the asynchronous PDO's a mean value is calculated based on the inhibit time of the asynchronous PDO.
Total Time/Cycle	Total time to transmit all CAN frames
Load	Bus load caused by all objects of this type

Clicking the button 'Show Figure' shows the following the timing diagram:

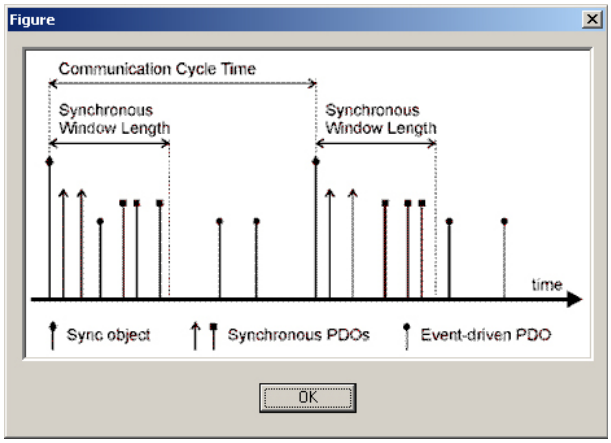


Figure 33: Cycle Time

6.3.2.3 Configuration View „Heartbeat Control“

The configuration view “Heartbeat Control” allows to define the error control behaviour of the master. Activate the heartbeat producer to monitor a breakdown of the master by the slave devices. Activate the heartbeat consumer to monitor a breakdown of a slave device.

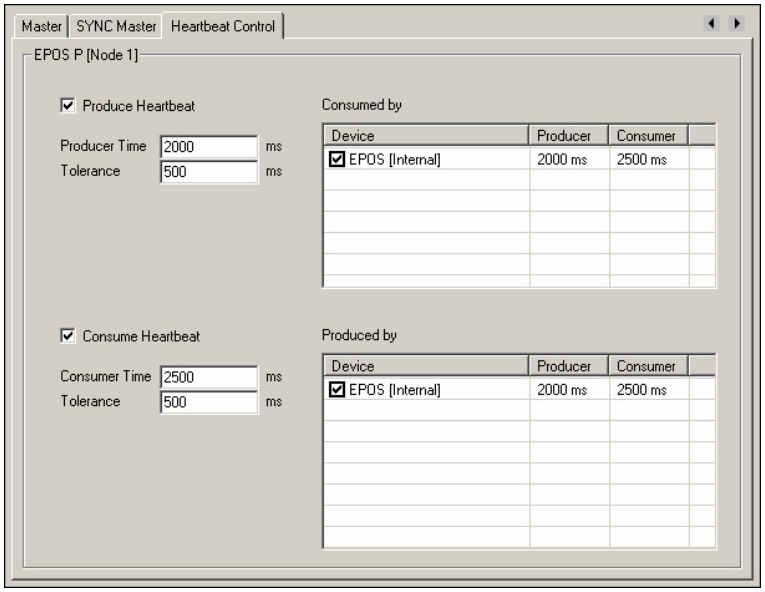


Figure 34: Configuration View ‘Heartbeat Control’

Option	Default	Description	
Producer Heartbeat	Disabled	Enable or disable the heartbeat producer	
Producer Time	2000 ms	Transmission rate of the heartbeat CAN frame	
Tolerance	500 ms	Tolerance time for the slave heartbeat consumer. The consumer time must always be higher than the producer time. A high bus load can delay the transmission of a heartbeat CAN frame.	
Consumed by	Disabled	Device	In case of a breakdown of the master (heartbeat producer), this device is going to error state.
		Producer	Heartbeat producer time
		Consumer	Heartbeat consumer time

Option	Default	Description	
Consumer Heartbeat	Disabled	Enable or disable the heartbeat consumer	
Consumer Time	2000 ms	Expected transmission rate of the heartbeat CAN frame	
Tolerance	500 ms	Tolerance time for the master heartbeat consumer. The consumer time must always be higher than the producer time. A high bus load can delay the transmission of a heartbeat CAN frame.	
Produced by	Disabled	Device	In case of a breakdown of the master (heartbeat consumer), this device is going to error state.
		Producer	Heartbeat producer time
		Consumer	Heartbeat consumer time

6.3.3 Slave Configuration

For the slave configuration, select the network and one of the slave items in the device selection.

6.3.3.1 Configuration View “Slave”

The configuration view “Slave” allows defining the behaviour of the slave device.

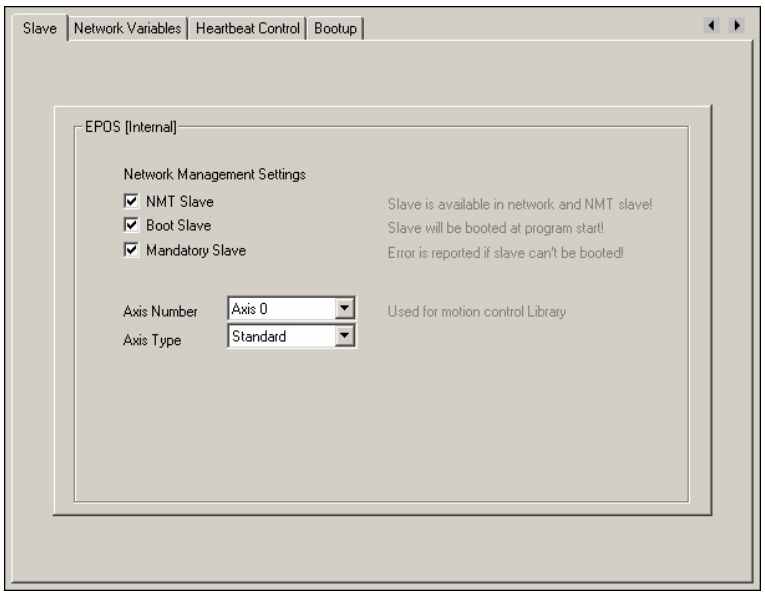


Figure 35: Configuration View ‘Slave’

Option	Default	Description
NMT Slave	Checked	The slave is available in CAN network as a NMT slave
Boot Slave	Checked	The slave will be booted at the program start
Mandatory Slave	Checked	Error is reported if slave can’t be booted
Axis Number	Axis X	Axis Number is used by all motion control function blocks. The default value is defined by the node Id. **Remark:** If no axis number is defined, the motion control function blocks can’t be used.
Axis Type	Standard	Axis Type is used by all motion control function blocks. **Remark:** If the axis type is not defined as ‘Standard’, the motion control function blocks can’t be used.

6.3.3.2 Tab “Network Variables”

The configuration view “Network Variables” allows to setup network variables for the IEC-61131 program.

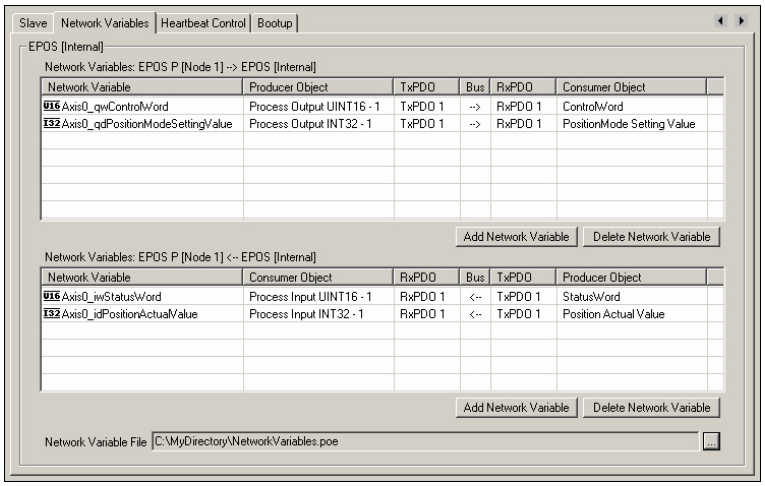


Figure 36: Configuration View ‘Network Variables’

Table Network Variables: EPOS P [Node 1] → EPOS [Internal]
Displays all configured network variables sent from the master to the slave.

Column	Description
Network Variable	Name of network variable to be used in IEC-61131 program. The network variables can be exported to a network variable file (*.poe)
Producer Object	Object in object dictionary of the master. This object is mapped to the transmit PDO
TxPDO	Configured transmit PDO to send data to the slave
Bus	Direction of the data exchange
RxPDO	Configured receive PDO to receive data from the master
Consumer Object	Object in object dictionary of the slave. This object is mapped to the receive PDO

Table Network Variables: EPOS P [Node 1] ← EPOS [Internal]
Displays all configured network variables sent from the slave to the master.

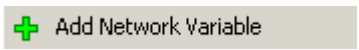
Column	Description
Network Variable	Name of network variable to be used in IEC-61131 program. The network variables can be exported to a network variable file (*.poe)
Consumer Object	Object in object dictionary of the master. This object is mapped to the receive PDO
RxPDO	Configured receive PDO to receive data from the slave
Bus	Direction of the data exchange
TxPDO	Configured transmit PDO to send data to the master
Producer Object	Object in object dictionary of the slave. This object is mapped to the transmit PDO

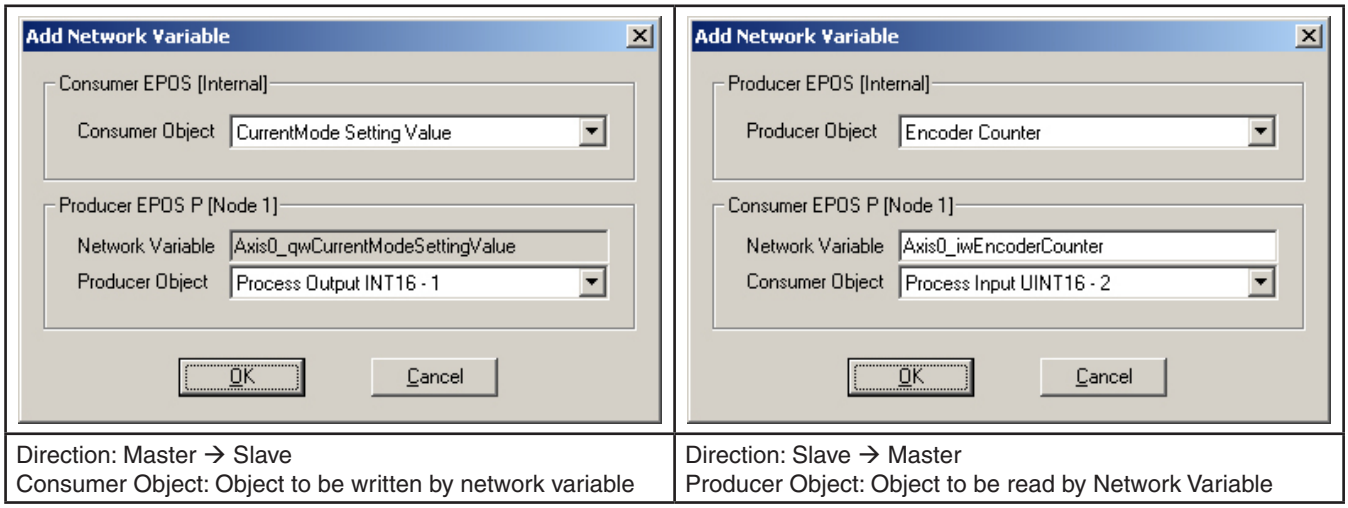
Add Network Variable:

A network variable can be added by:

⇨ clicking the button 

or

⇨ via the context menu (right mouse click) 



Direction: Master → Slave Consumer Object: Object to be written by network variable	Direction: Slave → Master Producer Object: Object to be read by Network Variable

Figure 37: Add Network Variable

Direction: Master → Slave

Parameter	Description
Consumer Object	Object to be written by network variable
Network Variable	Name of network variable to be used in IEC-61131 program
Producer Object	Process object of master

Direction: Slave → Master

Parameter	Description
Producer Object	Object to be read by network variable
Network Variable	Name of network variable to be used in IEC-61131 program
Consumer Object	Process object of master

Delete Network Variable:

A network variable can be deleted by selecting the network variable in the list and:

⇨ clicking the button 

or

⇨ via the context menu (right mouse click) 

Edit PDO Links:

The PDO links, automatically created by adding a network variable can be edited using the context menu item.

Context menu item (right mouse click): 

The dialogue ‘Edit PDO Links’ shows all PDO’s linked between the master and the slave device. The configuration of the PDO can be changed using this dialogue.

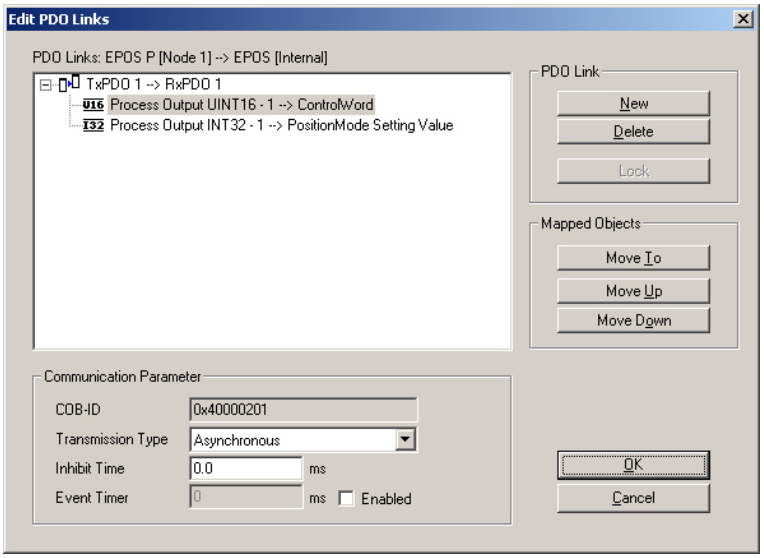


Figure 38: Edit PDO Links

Communication Parameter:

Parameter	Description	
COB-ID	COB-ID of the linked PDO's.	
Transmission Type	Synchronous	The PDO transmission is triggered by the Sync Master
	Asynchronous	The PDO transmission is trigger by a value change or an event timer
	Asynchronous RTR only	Do not use for network variables!
Inhibit Time	Minimal transmission interval for asynchronous PDO's **Remark:** An inhibit time of zero is a potential risk for a bus overload!	
Event Timer	The asynchronous PDO transmission is triggered by an elapsed event timer	

PDO Link:

Button	Description
New	Create a new PDO link between the master and slave devices
Delete	Delete an existing PDO link between the master and slave device. Only an empty PDO link can be deleted. Remove first the mapped objects.
Lock / Unlock	Lock or unlock a PDO link. A locked PDO can not be used by any other network variable.

Mapped Objects:

Button	Description
Move To	Move the selected objects to another PDO link
Move Up	Move the selected objects up in the list of mapped objects
Move Down	Move the selected object down in the list of mapped objects

Lock & Unlock PDO's:

Any PDO's of the master or slave devices can be locked or unlocked. A locked PDO can't be used by any other network variables.

Context menu item (right mouse click): 

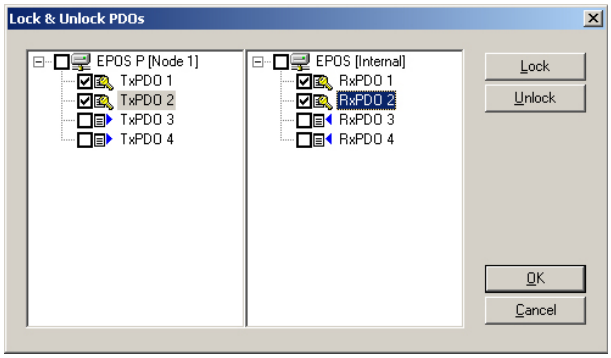


Figure 39: Lock & Unlock PDO's

Icons	Description
	Locked PDO. Can't be used by any other network variables
	Unlocked transmit PDO. Can be used by new network variables
	Unlocked receive PDO. Can be used by new network variables

Reset PDOs:

To create a good starting point for a network variable definition, the PDO configuration can be reset.

Context menu item (right mouse click): 

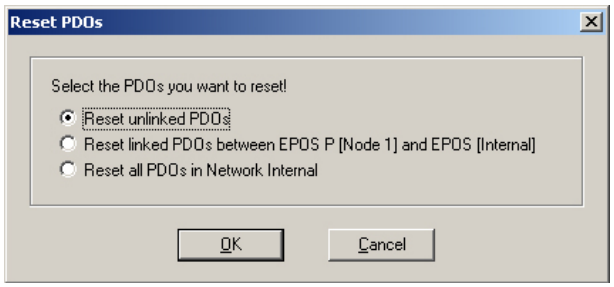
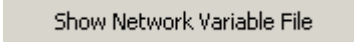


Figure 40: Reset PDO's

Options	Description
Reset unlinked PDO's	All active PDO's not linked to any known devices in the network will be deactivated. Inactive PDO's are then available for new network variables.
Reset linked PDO's between EPOS P and EPOS	All active and linked PDO's between two devices are reset. Use this option to clear the PDO configuration of two devices. All network variables are deleted.
Reset all PDO's in network	All active PDO's in a network are reset.

Show Network Variable File:

The declaration of the network variables for the IEC-61131 program are shown.

Context menu item (right mouse click): 

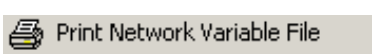
Save Network Variable File:

The declarations of the network variables for the IEC-61131 program are saved to a file (*.poe). This file can be included in a IEC-61131 program.

Context menu item (right mouse click): 

Print Network Variable File :

The declarations of the network variables for the IEC-61131 program are printed.

Context menu item (right mouse click): 

```
VAR_GLOBAL
    Axis0_iwStatusWord                      AT %IW64.0: UINT;
    Axis0_idPositionActualValue             AT %ID96.0: DINT;

    Axis0_qwControlWord                     AT %QW64.0: UINT;
    Axis0_qdPositionModeSettingValue        AT %QD96.0: DINT;

END_VAR
```

Figure 41: Declaration of network variables

6.3.3.3 Configuration View “Heartbeat Control”

The configuration view “Heartbeat Control” allows defining the error control behaviour of a slave device. Activate the heartbeat producer to monitor a breakdown of the slave by any other devices. Activate the heartbeat consumer to monitor a breakdown of any other device.

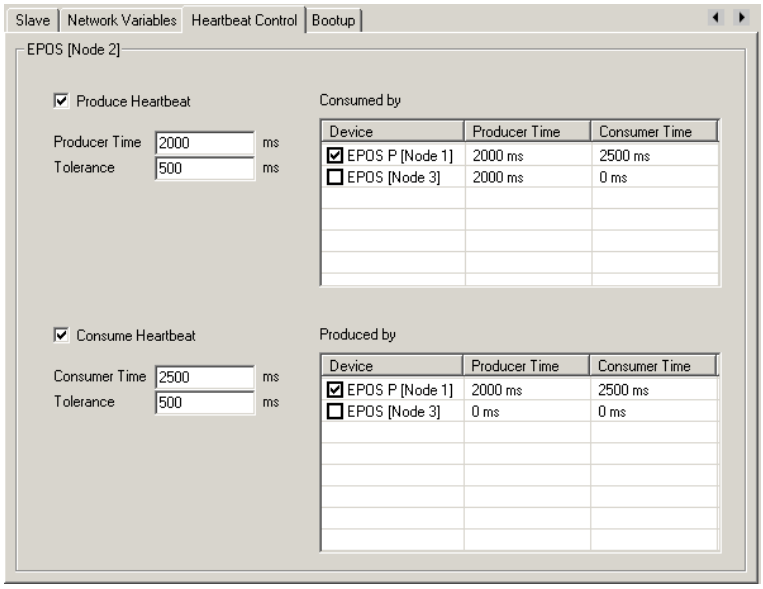


Figure 42: Configuration View “Heartbeat Control”

Option	Default	Description	
Producer Heartbeat	Disabled	Enable or disable the heartbeat producer	
Producer Time	2000 ms	Transmission rate of the heartbeat CAN frame	
Tolerance	500 ms	Tolerance time for the heartbeat consumer. The consumer time must always be higher than the producer time. A high bus load can delay the transmission of a heartbeat CAN frame.	
Consumed by	Disabled	Device	In case of a breakdown of the heartbeat producer, this device is going to error state.
		Producer	Heartbeat Producer Time
		Consumer	Heartbeat Consumer Time

Option	Default	Description	
Consumer Heartbeat	Disabled	Enable or disable the heartbeat consumers	
Consumer Time	2000 ms	Expected transmission rate of the heartbeat CAN frame	
Tolerance	500 ms	Tolerance time for the heartbeat consumer. The consumer time must always be higher than the producer time. A high bus load can delay the transmission of a heartbeat CAN frame.	
Produced by	Disabled	Device	In case of a breakdown of the heartbeat consumer, this device is going to error state.
		Producer	Heartbeat Producer Time
		Consumer	Heartbeat Consumer Time

6.3.3.4 Configuration View “Bootup”

The configuration view “Bootup” allows to define various bootup configuration checks. During configuration the identification values of the slave device are stored in the master. During bootup procedure the master is checking if the correct slave device is connected to the CAN bus. If a bootup check fails the IEC-61131 program will not be started.

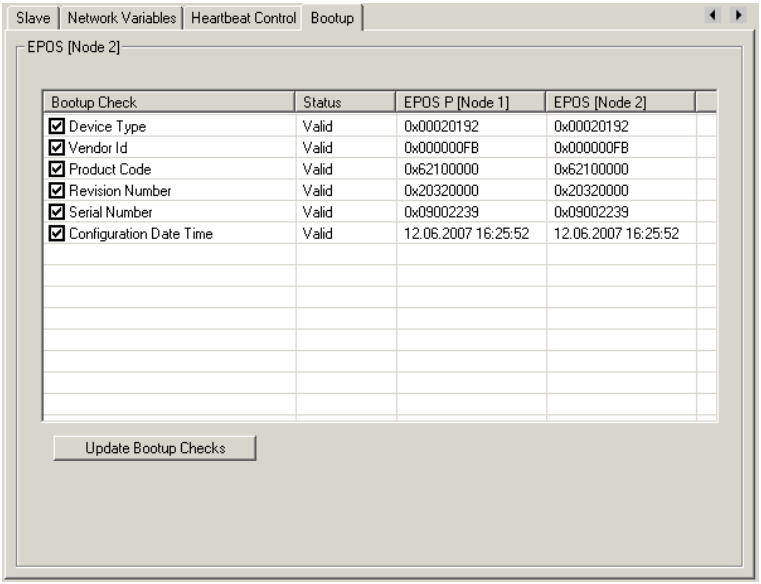


Figure 43: Configuration View “Bootup”

Bootup Check	Default	Description
Device Type	Disabled	Contains information about the device type. The lower 16-bit describes the CANopen device profile (i.e. 0x0192 = DSP 402)
Vendor Id	Disabled	Contains a unique value allocated to each manufacturer (i.e. 0x000000FB = maxon motor ag)
Product Code	Disabled	Contains a specific device version (i.e. 0x62100000 = Hardware Version EPOS 24/5)
Revision Number	Disabled	Contains a specific firmware version (i.e. 0x20320000 = Software Version EPOS 24/5)
Serial Number	Disabled	Contains a unique value allocated to each device (i.e. 0x62100000 = Hardware Version EPOS 24/5)
Configuration Date Time	Disabled	Contains information about the last change of the configuration settings.

6.3.4 Minimal Network Configuration

In order to use a motion control axis in a IEC-61131 program the following configuration steps have to be done.

Step 1: Create Project in EPOS Studio

1. Select menu item ‘New Project’ in the menu ‘File’
2. Select a EPOS P project template and click ‘Next’
3. Enter the project name, destination directory and click ‘Finish’

Step 2: Scan the Network Topology

1. Change to the tab ‘Communication’ in the navigation window
2. Select the icon for the CAN network and execute the command ‘Scanning Devices’ in the context menu
3. Setup the scanning settings

4. Start Scanning
5. Close the dialog ‘Scanning Devices’ clicking OK
6. Connect all new scanned devices

Step 3: Open the Tool ‘Network Configuration’

1. Change to the tab ‘Tools’ in the navigation window
2. Select the device EPOS P in device selection
3. Open the tool by clicking the item ‘Network Configuration

Step 4: Minimal Master Configuration

1. Select the master device EPOS P in the device selection
2. Select the configuration view ‘Master’ and configure the following options
   - NMT Master: Enabled
   - Start NMT Master: Enabled
   - Start NMT Slaves: Enabled
   - Boot Time: 500 ms
   - Start All NMT Slaves together: Enabled
3. Select the configuration view ‘SYNC Master’ and disable the Sync Producer
4. Select the configuration view ‘Heartbeat Control’ and disable the Heartbeat Producer

Step 5: Minimal Slave Configuration

1. Select one of the slave devices in the device selection
2. Select the configuration view ‘Slave’ and configure the following options
   - NMT Slave: Enabled
   - Boot Slave: Enabled
   - Mandatory Slave: Enabled
   - Axis Number: Select the axis number for example corresponding to the Node Id
   - Axis Type: Standard
3. Select the configuration view ‘Heartbeat Control’ and disable the Heartbeat Producer
4. Select the configuration view ‘Booting’ and disable all bootup checks
5. Repeat the slave configuration for all slaves in your system

Step 6: Save Network Configuration

1. Write and save the network configuration by clicking the OK button

Step 7: Start writing your IEC-61131 program

1. Open the programming tool and start writing your program addressing network devices

6.3.4.1 Communication via Function Blocks

In order to address network devices using motion control function blocks, all devices need a unique axis number. Executing the minimal network configuration for all devices. The devices can be addressed without any further configuration steps.

Motion Control Function Blocks:

Function Block		Configuration	
Parameter	AXIS_REF.AxisNo = Axis Number	Parameter	Axis Number
Function Block Example:		Configuration View 'Slave':	

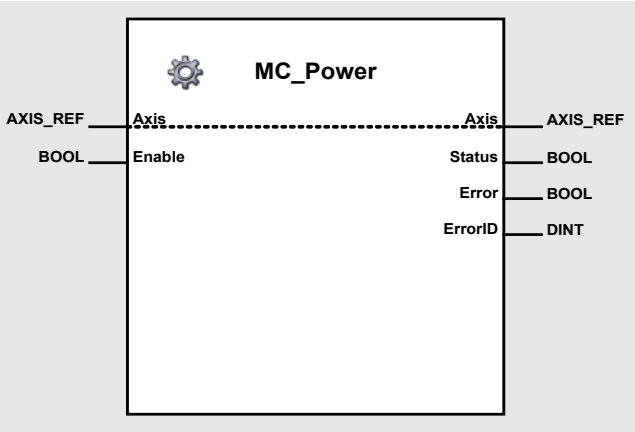
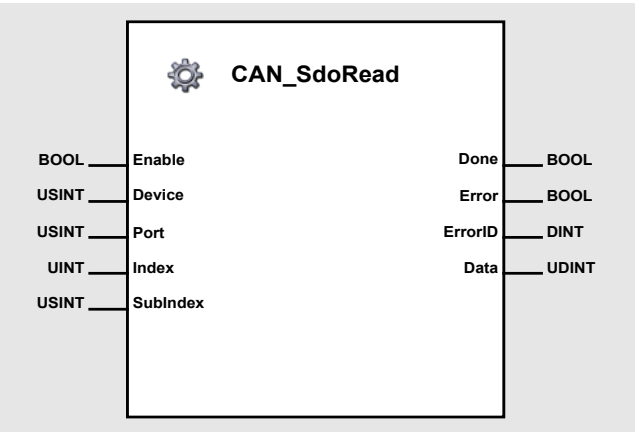
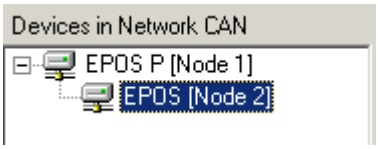


Figure 44: Motion control function block: Configuration of axis number

CANopen DS-301 Function Blocks:

Function Block		Configuration	
Parameter	Device: Node Id Port: 0 internal port, 1 CAN port	Parameter	Node Id
Function Block Example:		Device Selection:	





Can be changed by DIP switch or Startup Wizard

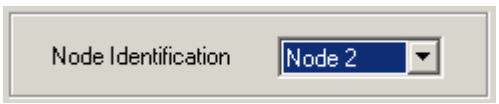


Figure 45: CANopen DS-301 function block: Configuration of node ID

6.3.4.2 Communication via Network Variables

In order to address network devices using network variables some additional configuration steps are necessary.

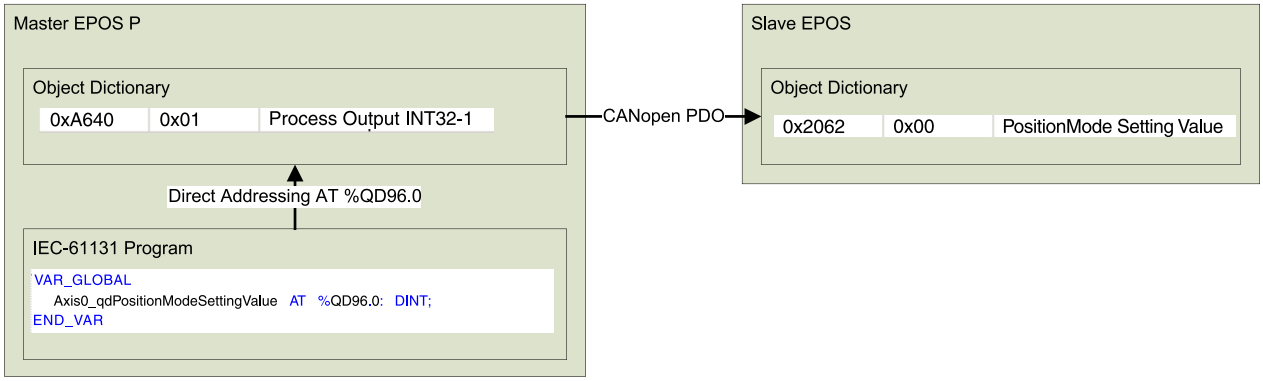
Step 1: Open Configuration View 'Network Variables'

1. Open the tool 'Network Configuration'
2. Select one of the slave devices in the device selection and activate the configuration view 'Network Variables'

Step 2: Define Output Network Variables
Network Variables from the master to the slave can be used to control a slave device.

1. Click the button 'Add Network Variable' in the upper part of the view
2. Select a consumer object in the selection combo box
3. Confirm selection by clicking OK
4. Repeat steps for each network variable

Network Variable from IEC-61131 Program to Slave



Configuration View 'Network Variables'

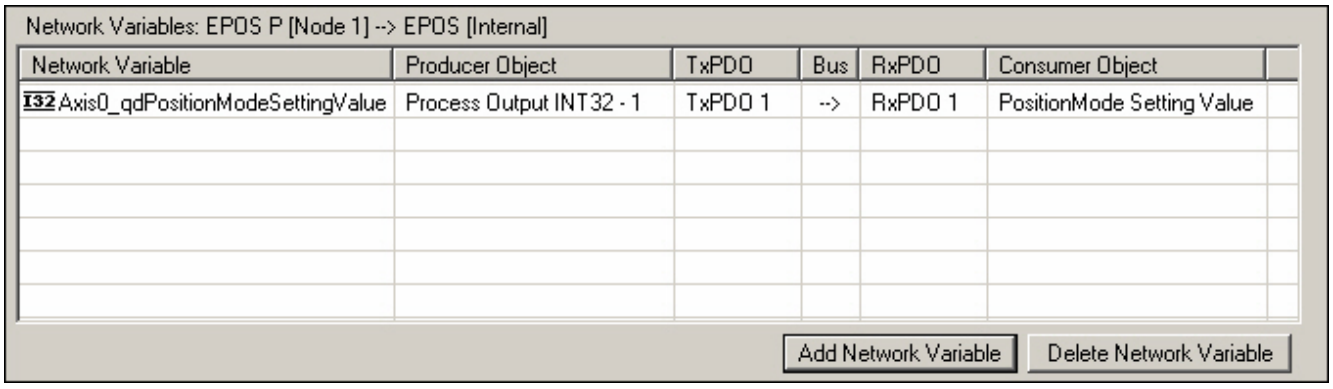
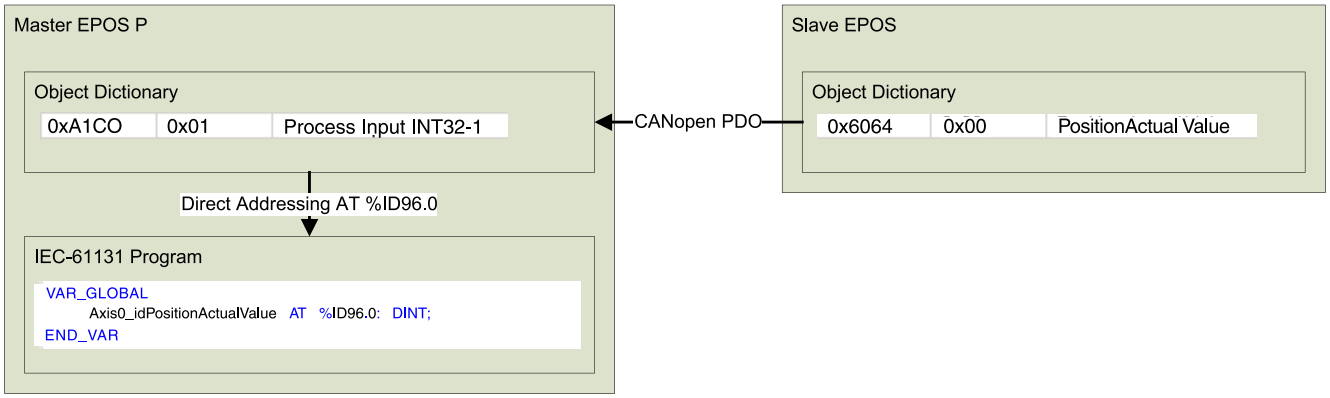


Figure 46: Output Network Variables

Step 3: Define Input Network Variables
Network Variables from the slave to the master can be used to monitor actual values.

1. Click the button 'Add Network Variable' in the lower part of the view
2. Select a producer object in the selection combo box
3. Confirm selection by clicking OK
4. Repeat steps for each network variable

Network Variable from Slave to IEC-61131 Program



Configuration View 'Network Variables'

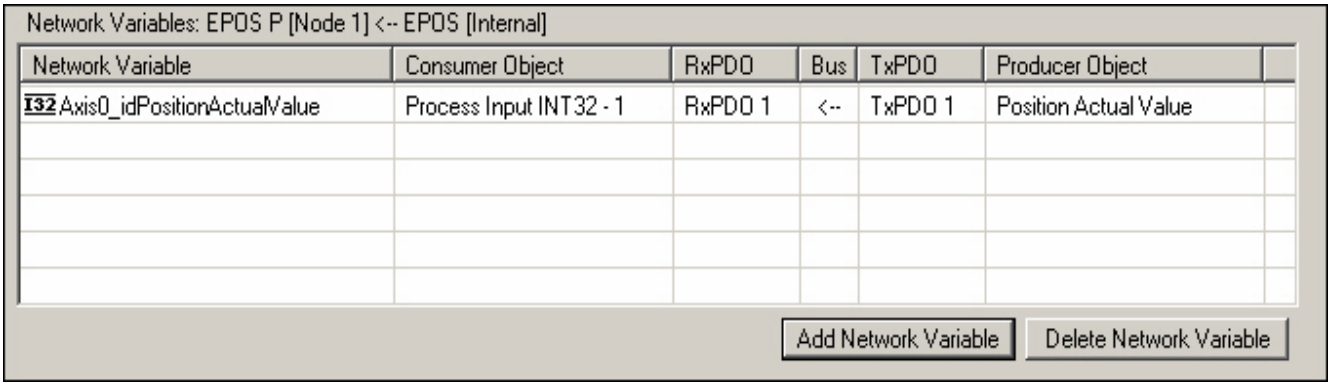


Figure 47: Input Network Variables

Step 4: Network Variable File (*.poe)

1. Click the browse button on the bottom of the view 
2. Enter a file name for the network variable export and close the dialogue

Figure 48: Network Variable File

Step 5: Save Network Configuration and Export Network Variables

1. Click the OK button to save the network configuration. The network variables are exported to the selected network variable file.

Step 6: Import Network Variables to IEC-61131 program

1. Open your IEC-61131 program in the programming tool 'Open PCS'.
2. Select the menu item 'Import' in the submenu 'File' of the menu 'File'
3. Click the context menu item 'Link to Active Resource' to use the network variables

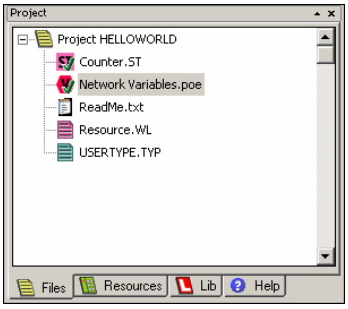


Figure 49: Project Browser in Programming Tool

7 Function Block Description

7.1 Motion Control Function Blocks

7.1.1 Function Block MC_Power

The Function Block MC_Power controls the power stage of the axis (enabled or disabled).

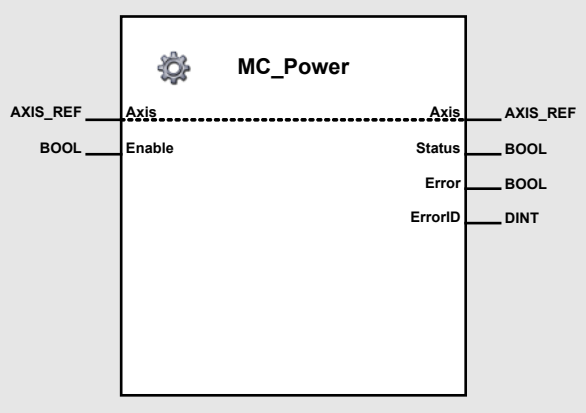


Figure 50: MC_Power

Input/Output Variable

Name	Data Type	Elements	Element Type	Default Value	Value Range
Axis	AXIS_REF	AxisNo	USINT	0	0 to 31

The input/output variable Axis defines the addressed axis. With the “Network Configuration” tool the axis number of the internal and the external axis can be set freely within the allowed value range.

Input Variable

Name	Data Type	Default Value	Value Range	Unit
Enable	BOOL	FALSE	TRUE, FALSE	-

As long as the Enable input is TRUE (positive state), the power stage of the axis is activated.

Output Variables

Name	Data Type	Default Value	Value Range	Unit
Status	BOOL	FALSE	FALSE, TRUE	-
Error	BOOL	FALSE	FALSE, TRUE	-
ErrorID	DINT	0	See chapter 10.2 Motion Control Functionblocks Error Codes	-

The output Status is showing the state of the power stage. An error during execution of the function block is signalled at the output Error. Additionally an ErrorID allows to get more information about the error cause.

Function Block Call using the programming language ST

```
-----------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis  : AXIS_REF := (AxisNo := 0);
fbPower : MC_Power; (* fbPower is instance of MC_Power *)

END_VAR
-----------------------------------------------------------------------------------------
(* Call function block instance *)
fbPower(Axis := myAxis, Enable := TRUE);
```

Important

The function block MC_Power has to be called until the output Status has the same value as the input Enable. Breaking this rule causes system errors.

7.1.2 Function Block MC_Reset

The Function Block MC_Reset is resetting all internal axis-related errors.

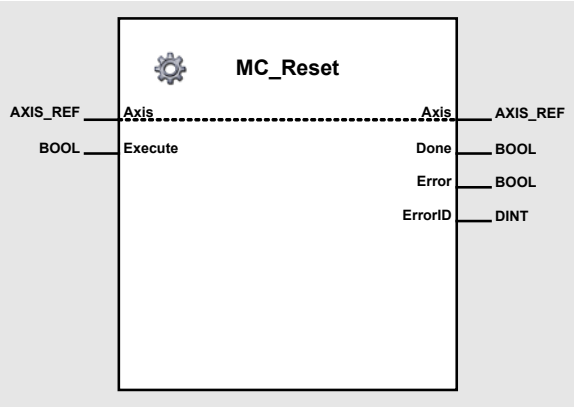


Figure 51: MC_Reset

Input/Output Variable

Name	Data Type	Elements	Element Type	Default Value	Value Range
Axis	AXIS_REF	AxisNo	USINT	0	0 to 31

The input/output variable Axis defines the addressed axis. With the “Network Configuration” tool the axis number of the internal and the external axis can be set freely within the allowed value range.

Input Variable

Name	Data Type	Default Value	Value Range	Unit
Execute	BOOL	FALSE	TRUE, FALSE	-

At a positive edge of the input Execute the axis status is changed from Errorstop to StandStill. After execution of MC_Reset the power stage has to be reenabled (MC_Power).

Output Variables

Name	Data Type	Default Value	Value Range	Unit
Done	BOOL	FALSE	FALSE, TRUE	-
Error	BOOL	FALSE	FALSE, TRUE	-
ErrorID	DINT	0	See chapter 10.2 Motion Control Functionblocks Error Codes	-

A successful reset of the axis status is signalled by the output Done. An error during execution of the function block is signalled at the output Error. Additionally an ErrorID allows to get more information about the error cause.

Function Block Call using the programming language ST

```
-----------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis  : AXIS_REF := (AxisNo := 0);
fbReset : MC_Reset; (* fbReset is instance of MC_Reset *)

END_VAR
-----------------------------------------------------------------------------------------
(* Call function block instance *)
fbReset(Axis := myAxis, Execute := TRUE);
```

Important
The function block MC_Reset has to be called until the termination is signalled at the output (Done or Error).

7.1.3 Function Block MC_ReadStatus

The Function Block MC_ReadStatus returns in detail the status of the axis with respect to the motion currently in progress.

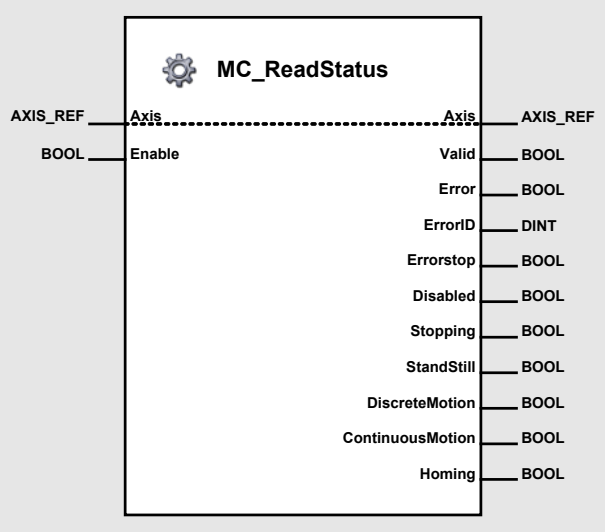


Figure 52: MC_ReadStatus

Input/Output Variable

Name	Data Type	Elements	Element Type	Default Value	Value Range
Axis	AXIS_REF	AxisNo	USINT	0	0 to 31

The input/output variable Axis defines the addressed axis. With the “Network Configuration” tool the axis number of the internal and the external axis can be set freely within the allowed value range.

Input Variable

Name	Data Type	Default Value	Value Range	Unit
Enable	BOOL	FALSE	TRUE, FALSE	-

As long as the Enable input is TRUE (positive state), the value of the status parameter is read continuously.

Output Variables

Name	Data Type	Default Value	Value Range	Unit
Valid	BOOL	FALSE	FALSE, TRUE	-
Error	BOOL	FALSE	FALSE, TRUE	-
ErrorID	DINT	0	See chapter 10.2 Motion Control Functionblocks Error Codes	-
Errorstop	BOOL	FALSE	FALSE, TRUE	-
Disabled	BOOL	FALSE	FALSE, TRUE	-
Homing	BOOL	FALSE	FALSE, TRUE	-
Stopping	BOOL	FALSE	FALSE, TRUE	-
StandStill	BOOL	FALSE	FALSE, TRUE	-
DiscreteMotion	BOOL	FALSE	FALSE, TRUE	-
ContinuousMotion	BOOL	FALSE	FALSE, TRUE	-

A successful update of the axis status is signalled by a positive state (TRUE) of the output Valid. An error during execution of the function block is signalled at the output Error. Additionally an ErrorID allows to get more information about the error cause.

The figure on the next page shows the possible states.

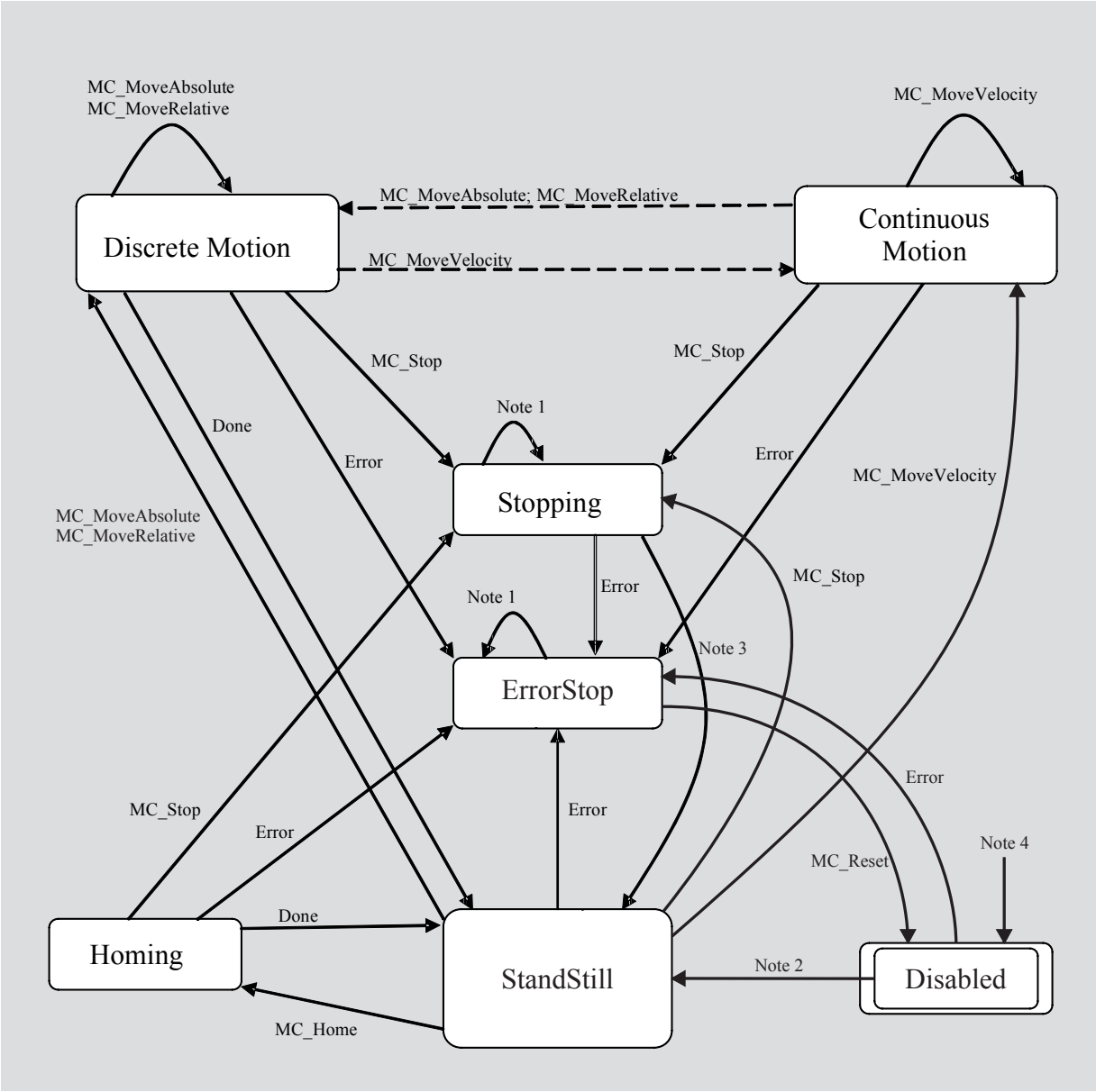


Figure 53: ReadStatus, possible States

Note 1: In this state Errorstop or Stopping, all Function Blocks can be called, although they will not be executed, except MC_Reset and Error - they will generate the transition to StandStill or Errorstop respectively.
Note 2: Power.Enable = TRUE and there is no error in the axis
Note 3: MC_Stop.Done
Note 4: MC_Power.Enable = FALSE

Function Block Call using the programming language ST

```
-----------------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis  : AXIS_REF := (AxisNo := 0);
fbRead  : MC_ReadStatus; (* fbRead is instance of MC_ReadStatus *)

END_VAR
-----------------------------------------------------------------------------------------------
(* Call function block instance *)
fbRead(Axis := myAxis, Enable := TRUE);
IF fbRead.Valid & fbRead.Errorstop THEN
  ...
END_IF;
```

7.1.4 Function Block MC_MoveAbsolute

The Function Block MC_MoveAbsolute commands a controlled motion to a specified absolute position. This motion is using a trapezoidal or sinusoidal profile.

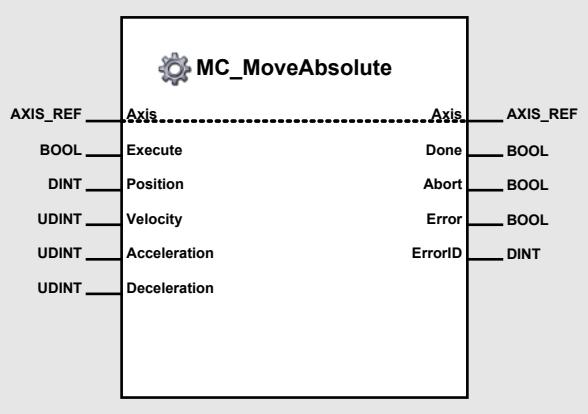


Figure 54: MC_MoveAbsolute

Input-/Output Variable

Name	Data Type	Elements	Element Type	Default Value	Value Range
Axis	AXIS_REF	AxisNo	USINT	0	0 to 31

The input/output variable Axis defines the addressed axis. With the “Network Configuration” tool the axis number of the internal and the external axis can be set freely within the allowed value range.

Input Variables

Name	Data Type	Default Value	Value Range	Unit
Execute	BOOL	FALSE	TRUE, FALSE	-
Position	DINT	0	- 2‘147‘483‘648 to + 2‘147‘483’647	qc = increments of the encoder
Velocity	UDINT	0	0 to 25‘000	rpm
Acceleration	UDINT	0	0 to 4‘294‘967’295	rpm/s
Deceleration	UDINT	0	0 to 4‘294‘967’295	rpm/s

A positive edge of the input signal Execute triggers a new absolute movement. The movement is using a profile corresponding to the values Velocity, Acceleration and Deceleration. The variable Position is defined in the unit quadcount [qc].

Output Variables

Name	Data Type	Default Value	Value Range	Unit
Done	BOOL	FALSE	FALSE, TRUE	-
Abort	BOOL	FALSE	FALSE, TRUE	-
Error	BOOL	FALSE	FALSE, TRUE	-
ErrorID	DINT	0	See chapter 10.2 Motion Control Functionblocks Error Codes	-

A successful positioning is signalled with a positive value (TRUE) at the output Done. If another function block instance is executing a movement using the same axis, the execution of this function block instance is immediately stopped. In this case a positive state (TRUE) at the output Abort is set. An error during the movement is signalled (TRUE) using the output Error. Additionally an ErrorID allows to get more information about the error cause.

The outputs Done, Abort and Error are reset with a negative state (FALSE) at the input Execute. If the input Execute is reset before the end of the positioning, the outputs Done, Abort and Error are showing the status of the positioning during one cycle. After one cycle the outputs are reset to negative state (FALSE).

The parameter values Velocity, Acceleration, Deceleration must only be defined at the first call of the function block. Calling the same function block for one more time the values don't have to be defined. In this case the values of the first call are used.

The figure below shows two cases of a calling sequence. The first sequence shows two complete movements. The second function block instance is started after the complete termination of the first movement. The second sequence shows an interrupted movement. Setting the variable Test triggers the start of the second function block instance during execution of the first one.

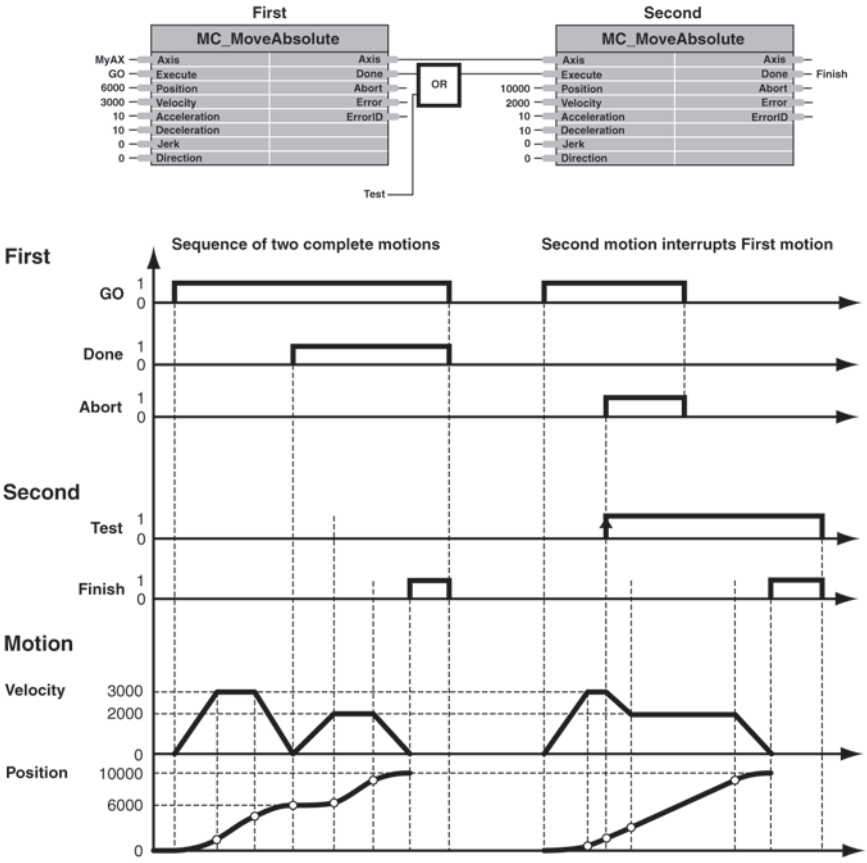


Figure 55: Move Absolute Sequence

Function Block Call using the programming language ST

```
-----------------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbMove  : MC_MoveAbsolute; (* fbMove is instance of MC_MoveAbsolute *)
Start   : BOOL := FALSE;
Pos     : DINT := 10000;

END_VAR
-----------------------------------------------------------------------------------------------
(* Call function block instance *)
fbMove(Axis:=myAxis,Execute:=Start,Position:=Pos,Velocity:=25,Acceleration:=50,Deceleration:=50);
```

Important

The execution of a function block instance might take longer than one PLC cycle. For a proper working system a function block instance has to be called (Execute or Enable) at every program cycle until its termination is signalled by the outputs Done, Error or Abort. At every call the function block instance will continue at its actual internal state where it has stopped during the previous PLC program cycle. Breaking this rule causes system errors. Especially if the function block uses CAN communication services which might not be finished fast enough.

7.1.5 Function Block MC_MoveRelative

The Function Block MC_MoveRelative commands a controlled motion of a specified distance relative to the actual position at the time of the execution. This movement is using a trapezoidal or sinusoidal profile. The new absolute target position is defined by the Distance added to the actual position.

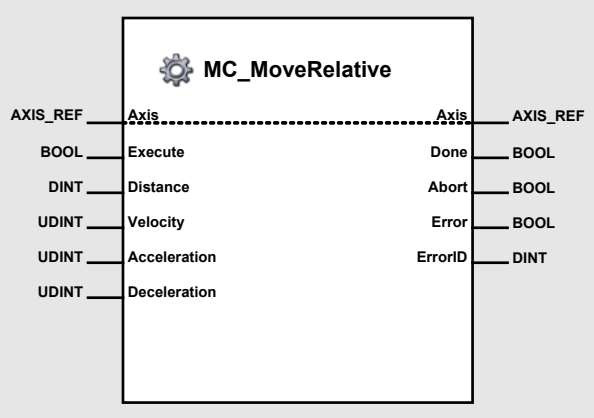


Figure 56: MC_MoveRelative

Input/Output Variable

Name	Data Type	Elements	Element Type	Default Value	Value Range
Axis	AXIS_REF	AxisNo	USINT	0	0 to 31

The input/output variable Axis defines the addressed axis. With the “Network Configuration” tool the axis number of the internal and the external axis can be set freely within the allowed value range.

Input Variables

Name	Data Type	Default Value	Value Range	Unit
Execute	BOOL	FALSE	TRUE, FALSE	-
Distance	DINT	0	- 2’147’483’648 to + 2’147’483’647	qc
Velocity	UDINT	0	0 to 25’000	rpm
Acceleration	UDINT	0	0 to 4’294’967’295	rpm/s
Deceleration	UDINT	0	0 to 4’294’967’295	rpm/s

A positive edge of the input signal Execute triggers a new relative movement. The defined distance is added to the actual position and commanded as a new absolute target position. The movement is using a profile corresponding to the values Velocity, Acceleration, Deceleration. The variable Distance is defined in the unit quadcount [qc].

Output Variables

Name	Data Type	Default Value	Value Range	Unit
Done	BOOL	FALSE	FALSE, TRUE	-
Abort	BOOL	FALSE	FALSE, TRUE	-
Error	BOOL	FALSE	FALSE, TRUE	-
ErrorID	DINT	0	See chapter 10.2 Motion Control Functionblocks Error Codes	-

A successful positioning is signalled with a positive value (TRUE) at the output Done. If another function block instance is executing a movement using the same axis, the execution of this function block instance is immediately stopped. In this case a positive state (TRUE) at the output Abort is set. An error during the movement is signalled (TRUE) using the output Error. Additionally an ErrorID allows to get more information about the error cause.

The outputs Done, Abort and Error are reset with a negative state (FALSE) at the input Execute. If the input Execute is reset before the end of the positioning, the outputs Done, Abort and Error are showing the status of the positioning during one cycle. After one cycle the outputs are reset to negative state (FALSE).
The parameter values Velocity, Acceleration, Deceleration must only be defined at the first call of the function block. Calling the same function block for one more time the values do not have to be defined. In this case the values of the first call are used.

The figure below shows two cases of a calling sequence. The first sequence shows two complete movements. The second function block instance is started after the complete termination of the first movement. The second sequence shows an interrupted movement. Setting the variable Test triggers the start of the second function block instance during execution of the first one.

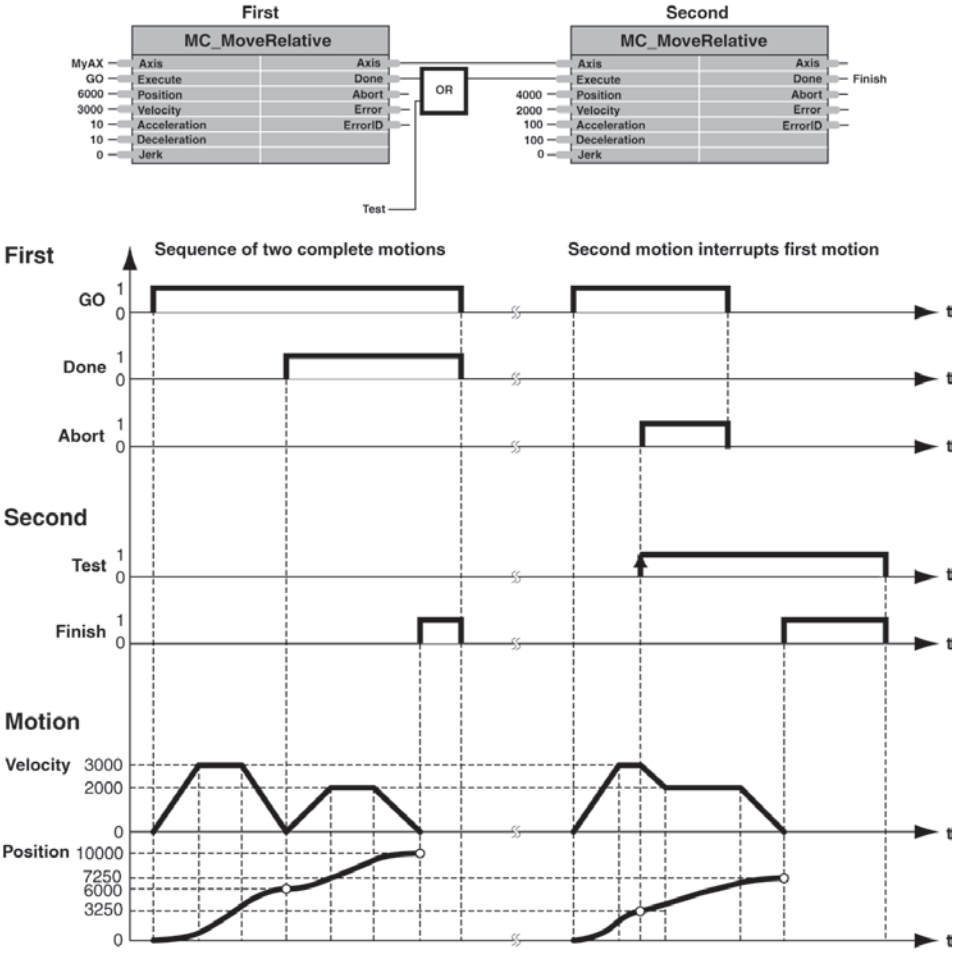


Figure 57: Move Relative Sequence

Function Block Call using the programming language ST

```
-----------------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis  : AXIS_REF := (AxisNo := 0);
fbMoveR : MC_MoveRelative; (* fbMove is instance of MC_MoveRelative *)
Start           : BOOL := FALSE;
Pos             : DINT := 10000;

END_VAR
-----------------------------------------------------------------------------------------------
(* Call function block instance *)
fbMoveR(Axis := myAxis, Execute := Start, Distance := Pos, Velocity := 1000,
Acceleration := 1000, Deceleration := 1000);
```

Important

The execution of a function block instance might take longer than one PLC cycle. For a proper working system a function block instance has to be called (Execute or Enable) at every program cycle until its termination is signalled by the outputs Done, Error or Abort. At every call the function block instance will continue at its actual internal state where it has stopped during the previous PLC program cycle. Breaking this rule causes system errors. Especially if the function block uses CAN communication services which might not be finished fast enough.

7.1.6 Function Block MC_MoveVelocity

The Function Block MC_MoveVelocity commands a never ending controlled motion at a specified velocity using the trapezoidal or the sinusoidal acceleration profile.

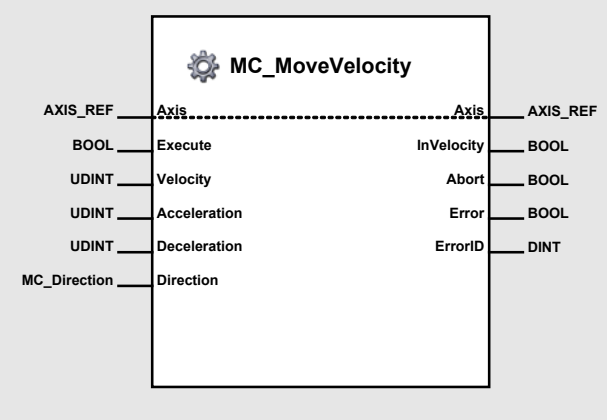


Figure 58: MC_MoveVelocity

Input/Output Variable

Name	Data Type	Elements	Element Type	Default Value	Value Range
Axis	AXIS_REF	AxisNo	USINT	0	0 to 31

The input/output variable Axis defines the addressed axis. With the “Network Configuration” tool the axis number of the internal and the external axis can be set freely within the allowed value range.

Input Variables

Name	Data Type	Default Value	Value Range	Unit
Execute	BOOL	FALSE	TRUE, FALSE	-
Velocity	UDINT	0	0 to 25’000	rpm
Acceleration	UDINT	0	0 to 4’294’967’295	rpm/s
Deceleration	UDINT	0	0 to 4’294’967’295	rpm/s
Direction	Enum MC_Direction	MCpositive	MCpositive MCnegative	-

A positive edge of the input signal Execute triggers a new velocity movement. This continuous movement is defined by the input variable Velocity. The function block MC_Stop must be used to stop the movement. Another call of MC_MoveVelocity changes the active velocity. The input variable Velocity has to be a positive value higher than 0. The movement direction is defined by the variable Direction. The velocity profile is executed using the variable values of Acceleration and Deceleration.

Output Variables

Name	Data Type	Default Value	Value Range	Unit
InVelocity	BOOL	FALSE	FALSE, TRUE	-
Abort	BOOL	FALSE	FALSE, TRUE	-
Error	BOOL	FALSE	FALSE, TRUE	-
ErrorID	DINT	0	See chapter 10.2 Motion Control Functionblocks Error Codes	-

Reaching the commanded velocity is signalled by the output InVelocity. If another function block instance is executing a movement using the same axis, the execution of this function block instance is immediately stopped. In this case a positive state (TRUE) at the output Abort is set. An error during the movement is signalled (TRUE) using the output Error. Additionally an ErrorID allows to get more information about the error cause.

The outputs InVelocity, Abort and Error are reset with a negative state (FALSE) at the input Execute. If the input Execute is reset before the end of the positioning, the outputs InVelocity, Abort and Error are showing the status of the positioning during one cycle. After one cycle the outputs are reset to negative state (FALSE).

The parameter values Acceleration, Deceleration, Direction must only be defined at the first call of the function block. Calling the same function block for one more time the values don't have to be defined. In this case the values of the first call are used.

The figure below shows two cases of a calling sequence. The first sequence shows two complete movements. The second function block instance is started after the complete termination of the first movement. The second sequence shows an interrupted movement. Setting the variable Test triggers the start of the second function block instance during execution of the first one.

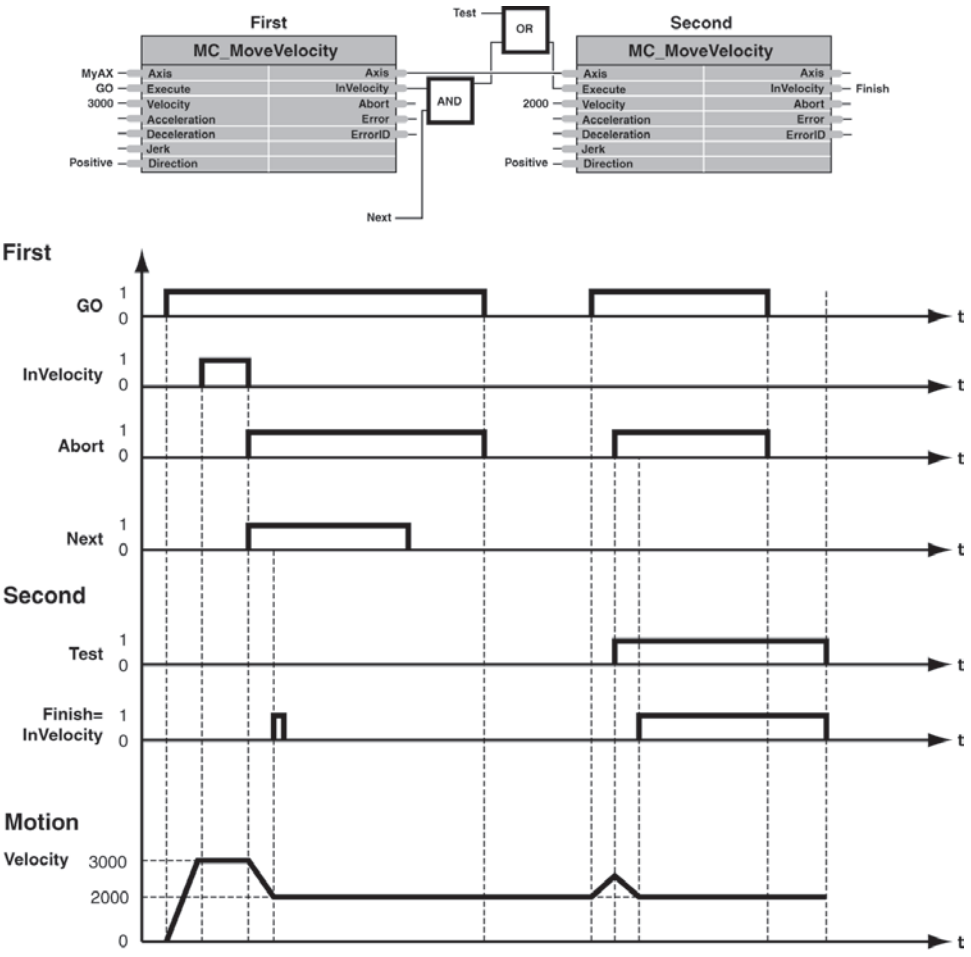


Figure 59: Move Velocity Sequence

Function Block Call using the programming language ST

```
-----------------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis  : AXIS_REF := (AxisNo := 0);
fbVelo          : MC_MoveVelocity; (* fbVelo is instance of MC_MoveVelocity *)
Start           : BOOL := FALSE;
END_VAR
-----------------------------------------------------------------------------------------------
(* Call function block instance *)
fbVelo(Axis := myAxis, Execute := Start, Velocity := 2000, Acceleration := 1000,
Deceleration := 1000, Direction := MCpositive);
```

Important

The execution of a function block instance might take longer than one PLC cycle. For a proper working system a function block instance has to be called (Execute or Enable) at every program cycle until its termination is signalled by the outputs Done, Error or Abort. At every call the function block instance will continue at its actual internal state where it has stopped during the previous PLC program cycle. Breaking this rule causes system errors. Especially if the function block uses CAN communication services which might not be finished fast enough.

7.1.7 Function Block MC_Home

The Function Block MC_Home commands the axis to perform the homing procedure. The absolute home position is searched using one of the available homing methods (see EPOS Firmware Specification).

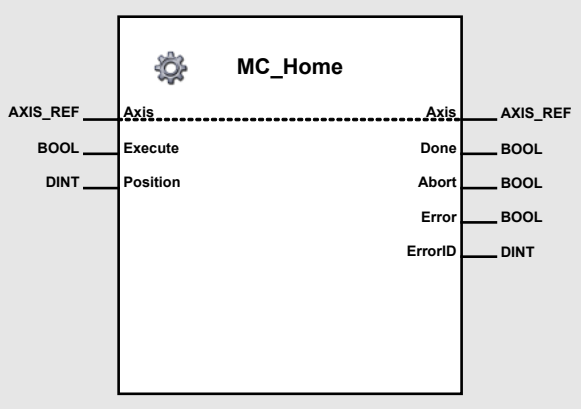


Figure 60: MC_Home

Input/Output Variable

Name	Data Type	Elements	Element Type	Default Value	Value Range
Axis	AXIS_REF	AxisNo	USINT	0	0 to 31

The input/output variable Axis defines the addressed axis. With the “Network Configuration” tool the axis number of the internal and the external axis can be set freely within the allowed value range.

Input Variables

Name	Data Type	Default Value	Value Range	Unit
Execute	BOOL	FALSE	TRUE, FALSE	-
Position	DINT	0	- 2’147’483’648 to + 2’147’483’647	qc

A positive edge of the input signal Execute triggers a new homing procedure. The input variable ‘Position’ defines the new home position value after a successful homing procedure. Additional parameters for a homing procedure have to be configured using the function block MC_WriteParameter. Have a look at the document EPOS Firmware Specification for further information about homing parameters.

Output Variables

Name	Data Type	Default Value	Value Range	Unit
Done	BOOL	FALSE	FALSE, TRUE	-
Abort	BOOL	FALSE	FALSE, TRUE	-
Error	BOOL	FALSE	FALSE, TRUE	-
ErrorID	DINT	0	See chapter 10.2 Motion Control Functionblocks Error Codes	-

A homing procedure has to be started when the axis is not moving. If the axis is moving at the start of the function block MC_Home an error is generated.

A successful terminated homing procedure is signalled by the output Done. If another function block instance is starting a homing procedure using the same axis, the execution of the first function block instance is immediately stopped. In this case a positive state (TRUE) at the output Abort is set. An error during the homing procedure is signalled (TRUE) using the output Error. Additionally the ErrorID allows to get more information about the error cause.

The outputs Done, Abort and Error are reset with a negative state (FALSE) at the input Execute. If the input Execute is reset before the end of the positioning the outputs Done, Abort and Error are showing the status of the positioning during one cycle. After one cycle the outputs are reset to negative state (FALSE).

The parameter value Position must be defined only at the first call of the function block. Calling the same function block for one more time the values don't have to be defined. In this case the values of the first call are used.

Function Block Call using the programming language ST

```
------------------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbHome  : MC_Home; (* fbHome is instance of MC_Home *)
Start   : BOOL := FALSE;

END_VAR
------------------------------------------------------------------------------------------------
(* Call function block instance *)
fbHome(Axis := myAxis, Execute := Start, Position := 0);
```

7.1.8 Function Block MC_Stop

The Function Block MC_Stop commands a controlled motion stop of the axis using a trapezoidal or sinusoidal deceleration profile.

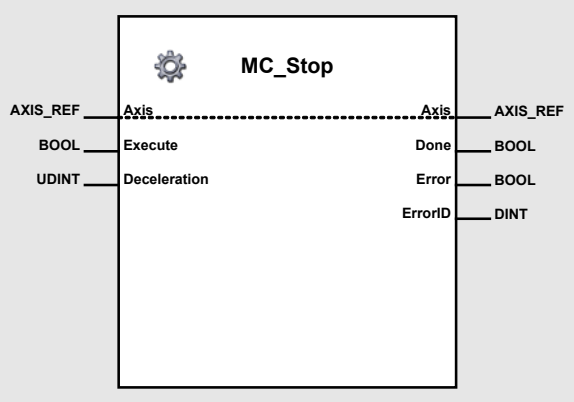


Figure 61: MC_Stop

Input/Output Variable

Name	Data Type	Elements	Element Type	Default Value	Value Range
Axis	AXIS_REF	AxisNo	USINT	0	0 to 31

The input/output variable Axis defines the addressed axis. With the “Network Configuration” tool the axis number of the internal and the external axis can be set freely within the allowed value range.

Input Variables

Name	Data Type	Default Value	Value Range	Unit
Execute	BOOL	FALSE	TRUE, FALSE	-
Deceleration	UDINT	0	0 to 4’294’967’295	rpm/s

A positive edge at the input Execute stops the axis using a defined deceleration profile.

Output Variables

Name	Data Type	Default Value	Value Range	Unit
Done	BOOL	FALSE	FALSE, TRUE	-
Error	BOOL	FALSE	FALSE, TRUE	-
ErrorID	DINT	0	See chapter 10.2 Motion Control Functionblocks Error Codes	-

Successfully stopping the axis is signalled by the output Done. An error during the deceleration profile is signalled (TRUE) using the output Error. Additionally an ErrorID allows to get more information about the error cause.

The outputs Done and Error are reset setting a negative state (FALSE) to the input Execute. If the input Execute is reset before the end of the positioning the outputs Done and Error are showing the status of the stopping during one cycle. After one cycle the outputs are reset to negative state (FALSE).

Function Block Call using the programming language ST

```
-----------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis  : AXIS_REF := (AxisNo := 0);
fbStop  : MC_Stop; (* fbStop is instance of MC_Stop *)
Start   : BOOL := FALSE;
END_VAR
-----------------------------------------------------------------------------------------
(* Call function block instance *)
fbStop(Axis := myAxis, Execute := Start, Deceleration := 1000);
```

7.1.9 Function Block MC_ReadParameter

The Function Block MC_ReadParameter returns an axis parameter value.

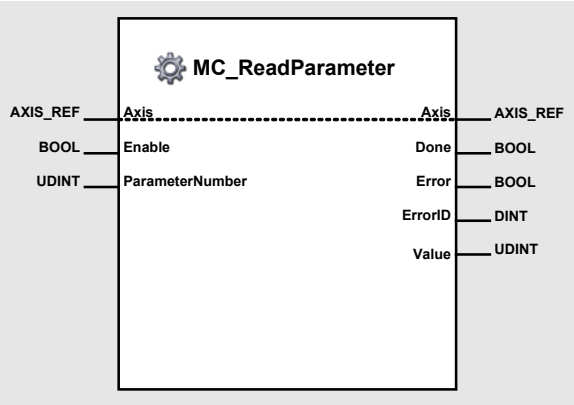


Figure 62: MC_ReadParameter

Input/Output Variable

Name	Data Type	Elements	Element Type	Default Value	Value Range
Axis	AXIS_REF	AxisNo	USINT	0	0 to 31

The input/output variable Axis defines the addressed axis. With the “Network Configuration” tool the axis number of the internal and the external axis can be set freely within the allowed value range.

Input Variables

Name	Data Type	Default Value	Value Range	Unit
Enable	BOOL	FALSE	TRUE, FALSE	-
ParameterNumber	UDINT	0	**PLCopen parameter:** 1 CommandedPosition 2 SWLimitPos 3 SWLimitNeg 7 MaxPositionLag 8 MaxVelocitySystem 9 MaxVelocityAppl 10 ActualVelocity 11 CommandedVelocity 13 MaxAccelerationAppl 15 MaxDecelerationAppl **CANopen objects:** 16#xxxxyyzz multiplexer (hex) xxxx: Object Index (hex) yy: Object Sub-index (hex) zz: Object Length (hex)	-

As long as the Enable input is TRUE (positive state), the value of a specified parameter is read continuously. The input ParameterNumber is defining the parameter to be read. Beside the listed parameter, CANopen objects can be read using the ParameterNumber as a multiplexer. So it’s possible to read all EPOS objects from the object dictionary (see document Firmware Specification).

The multiplexer is composed of 2 bytes object index (Byte 3 and 2), 1 byte object sub-index (Byte 1) and 1 byte object length (Byte 0).

Multiplexer Example

ParameterNumber = 16#207C0102

Name = Analog Input 1 of EPOS
Object Index = 16#207C
Object Sub-index = 16#01
Object Length = 16#02

Output Variables

Name	Data Type	Default Value	Value Range	Unit
Done	BOOL	FALSE	FALSE, TRUE	-
Error	BOOL	FALSE	FALSE, TRUE	-
ErrorID	DINT	0	See chapter 10.2 Motion Control Functionblocks Error Codes	-
Value	UDINT	0	0 to 4'294'967'295	-

A successful read operation is signalled by the output Done. The value can be read from the output Value. An error during execution of the function block is signalled at the output Error. Additionally an ErrorID allows to get more information about the error cause.

Function Block Call using the programming language ST

```
-----------------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis  : AXIS_REF := (AxisNo := 0);
fbReadP         : MC_ReadParameter; (* fbReadP is instance of MC_ReadParameter *)

END_VAR
-----------------------------------------------------------------------------------------------
(* Function Block call for updating the actual velocity *)
fbReadP(Axis := myAxis, Enable := TRUE, ParameterNumber := 10);

(* Function Block call for reading the CANopen object Analog Input 1*)
fbReadP(Axis := myAxis, Enable := TRUE, ParameterNumber := 16#207C0102);
```

Important
The execution of a function block instance might take longer than one PLC cycle. For a proper working system a function block instance has to be called (Execute or Enable) at every program cycle until its termination is signalled by the outputs Done, Error or Abort. At every call the function block instance will continue at its actual internal state where it has stopped during the previous PLC program cycle. Breaking this rule causes system errors. Especially if the function block uses CAN communication services which might not be finished fast enough.

7.1.10 Function Block MC_ReadBoolParameter

The Function Block MC_ReadBoolParameter returns an axis parameter value with datatype boolean.

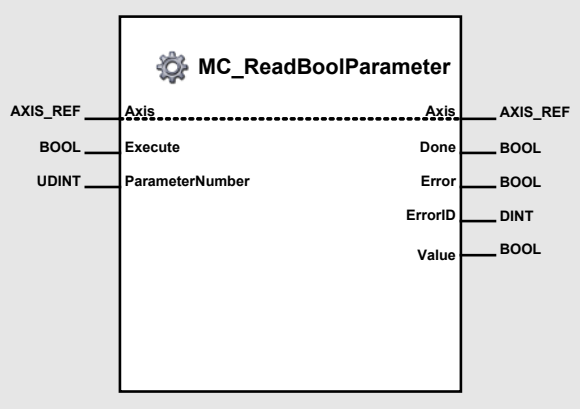


Figure 63: MC_ReadBoolParameter

Input/Output Variable

Name	Data Type	Elements	Element Type	Default Value	Value Range
Axis	AXIS_REF	AxisNo	USINT	0	0 to 31

The input/output variable Axis defines the addressed axis. With the “Network Configuration” tool the axis number of the internal and the external axis can be set freely within the allowed value range.

Input Variables

Name	Data Type	Default Value	Value Range	Unit
Enable	BOOL	FALSE	TRUE, FALSE	-
ParameterNumber	UDINT	0	4 EnableLimitPos 5 EnableLimitNeg 6 EnablePosLagMonitoring	-

As long as the Enable input is TRUE (positive state), the value of a specified boolean parameter is read continuously. The input ParameterNumber is defining the parameter to be read.

Output Variables

Name	Data Type	Default Value	Value Range	Unit
Done	BOOL	FALSE	FALSE, TRUE	-
Error	BOOL	FALSE	FALSE, TRUE	-
ErrorID	DINT	0	See chapter 10.2 Motion Control Functionblocks Error Codes	-
Value	BOOL	0	FALSE, TRUE	-

A successful read operation is signalled by the output Done. The value can be read from the output Value. An error during execution of the function block is signalled at the output Error. Additionally an ErrorID allows to get more information about the error cause.

Function Block Call using the programming language ST

```
-----------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis  : AXIS_REF := (AxisNo := 0);
fbReadB : MC_ReadBoolParameter; (* fbReadB is instance of MC_ReadBoolParameter *)
END_VAR
-----------------------------------------------------------------------------------------
(* Function Block Call for updating the actual velocity*)
fbReadB(Axis := myAxis, Enable := TRUE, ParameterNumber := 4);
```

Important

The execution of a function block instance might take longer than one PLC cycle. For a proper working system a function block instance has to be called (Execute or Enable) at every program cycle until its termination is signalled by the outputs Done, Error or Abort. At every call the function block instance will continue at its actual internal state where it has stopped during the previous PLC program cycle. Breaking this rule causes system errors. Especially if the function block uses CAN communication services which might not be finished fast enough.

7.1.11 Function Block MC_WriteParameter

The Function Block MC_WriteParameter modifies the value of an axis parameter.

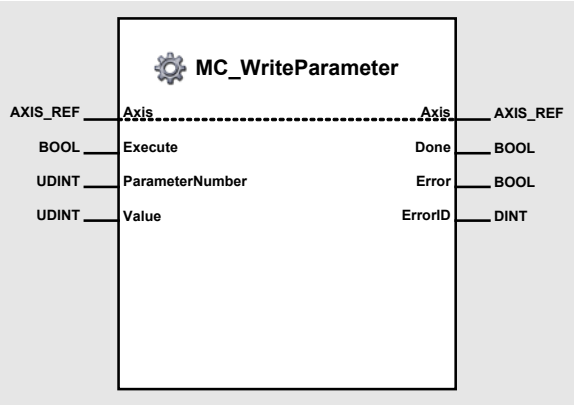


Figure 64: MC_WriteParameter

Input/Output Variable

Name	Data Type	Elements	Element Type	Default Value	Value Range
Axis	AXIS_REF	AxisNo	USINT	0	0 to 31

The input/output variable Axis defines the addressed axis. With the “Network Configuration” tool the axis number of the internal and the external axis can be set freely within the allowed value range.

Input Variables

Name	Data Type	Default Value	Value Range	Unit
Execute	BOOL	FALSE	TRUE, FALSE	-
ParameterNumber	UDINT	0	**PLCopen parameter:** 2 SWLimitPos 3 SWLimitNeg 7 MaxPositionLag 8 MaxVelocitySystem 9 MaxVelocityAppl 11 CommandedVelocity 13 MaxAccelerationAppl 15 MaxDecelerationAppl **CANopen objects:** 16#xxxxyyzz multiplexer (hex) xxxx: Object Index (hex) yy: Object Sub-index (hex) zz: Object Length (hex)	-
Value	UDINT	0	0 to 4’294’967’295	-

A positive edge at the input Execute is triggering a write operation of the specified parameter. The input ParameterNumber is defining the parameter to be written. Beside the listed parameter, CANopen objects can be written using the ParameterNumber as a multiplexer. So it is possible to write all EPOS objects to the object dictionary (see document Firmware Specification).

The multiplexer is composed of 2 bytes object index (Byte 3 and 2), 1 byte object sub-index (Byte 1) and 1 byte object length (Byte 0).

Multiplexer Example

ParameterNumber = 16#20780102

Name = Digital Outputs of EPOS
Object Index = 16#2078
Object Sub-index = 16#01
Object Length = 16#02

Output Variables

Name	Data Type	Default Value	Value Range	Unit
Done	BOOL	FALSE	FALSE, TRUE	-
Error	BOOL	FALSE	FALSE, TRUE	-
ErrorID	DINT	0	See chapter 10.2 Motion Control Functionblocks Error Codes	-

A successful write operation is signalled by the output Done. An error during execution of the function block is signalled at the output Error. Additionally an ErrorID allows to get more information about the error cause.

Function Block Call using the programming language ST

```
-----------------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis  : AXIS_REF := (AxisNo := 0);
fbWriteP       : MC_WriteParameter; (* fbWriteP is instance of MC_WriteParameter *)

END_VAR
-----------------------------------------------------------------------------------------------
(* Function Block Call for writing the maximum velocity *)
fbWriteP(Axis := myAxis, Execute := TRUE, ParameterNumber :=9, Value := 9000);
-----------------------------------------------------------------------------------------------
(* Function Block Call for writing the digital outputs of the EPOS *)
fbWriteP(Axis := myAxis, Execute := TRUE, ParameterNumber := 16#20780102 , Value := 0xFFFF);
```

Important
The execution of a function block instance might take longer than one PLC cycle. For a proper working system a function block instance has to be called (Execute or Enable) at every program cycle until its termination is signalled by the outputs Done, Error or Abort. At every call the function block instance will continue at its actual internal state where it has stopped during the previous PLC program cycle. Breaking this rule causes system errors. Especially if the function block uses CAN communication services which might not be finished fast enough.

7.1.12 Function Block MC_ReadActualPosition

The Function Block MC_ReadActualPosition returns the actual position of an axis.

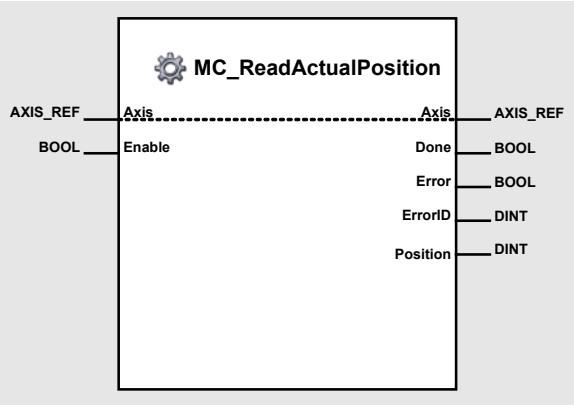


Figure 65: MC_ReadActualPosition

Input/Output Variable

Name	Data Type	Elements	Element Type	Default Value	Value Range
Axis	AXIS_REF	AxisNo	USINT	0	0 to 31

The input/output variable Axis defines the addressed axis. With the “Network Configuration” tool the axis number of the internal and the external axis can be set freely within the allowed value range.

Input Variable

Name	Data Type	Default Value	Value Range	Unit
Enable	BOOL	FALSE	TRUE, FALSE	-

As long as the Enable input is TRUE (positive state), the value of the actual position is read continuously.

Output Variables

Name	Data Type	Default Value	Value Range	Unit
Done	BOOL	FALSE	FALSE, TRUE	-
Error	BOOL	FALSE	FALSE, TRUE	-
ErrorID	DINT	0	See chapter 10.2 Motion Control Functionblocks Error Codes	-
Position	DINT	0	- 2’147’483’648 [min(DINT)] to + 2’147’483’647 [max(DINT)]	qc (Increments of the encoder)

A successful read operation is signalled by the output Done. The actual position can be read from the output Position. An error during execution of the function block is signalled at the output Error. Additionally an ErrorID allows to get more information about the error cause.

Function Block Call using the programming language ST

```
-----------------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis  : AXIS_REF := (AxisNo := 0);
fbPos           : MC_ReadActualPosition; (* fbPos is instance of MC_ReadActualPosition *)

END_VAR
-----------------------------------------------------------------------------------------------
(* Function Block Call for reading the actual position *)
fbPos(Axis := myAxis, Enable := TRUE);
```

Important

The execution of a function block instance might take longer than one PLC cycle. For a proper working system a function block instance has to be called (Execute or Enable) at every program cycle until its termination is signalled by the outputs Done, Error or Abort. At every call the function block instance will continue at its actual internal state where it has stopped during the previous PLC program cycle. Breaking this rule causes system errors. Especially if the function block uses CAN communication services which might not be finished fast enough.

7.1.13 Function Block MC_ReadActualVelocity

The Function Block MC_ReadActualVelocity returns the actual velocity of an axis.

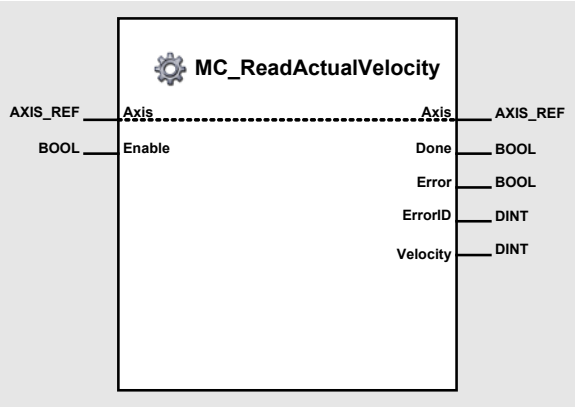


Figure 66: MC_ReadActualVelocity

Input/Output Variable

Name	Data Type	Elements	Element Type	Default Value	Value Range
Axis	AXIS_REF	AxisNo	USINT	0	0 to 31

The input/output variable Axis defines the addressed axis. With the “Network Configuration” tool the axis number of the internal and the external axis can be set freely within the allowed value range.

Input Variable

Name	Data Type	Default Value	Value Range	Unit
Enable	BOOL	FALSE	TRUE, FALSE	-

As long as the Enable input is TRUE (positive state), the value of the actual velocity is read continuously.

Output Variables

Name	Data Type	Default Value	Value Range	Unit
Done	BOOL	FALSE	FALSE, TRUE	-
Error	BOOL	FALSE	FALSE, TRUE	-
ErrorID	DINT	0	See chapter 10.2 Motion Control Functionblocks Error Codes	-
Velocity	DINT	0	- 2’147’483’648 [min(DINT)] to + 2’147’483’647 [max(DINT)]	rpm

A successful read operation is signalled by the output Done. The actual velocity can be read from the output Velocity. An error during execution of the function block is signalled at the output Error. Additionally an ErrorID allows to get more information about the error cause.

Function Block Call using the programming language ST

```
-----------------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbVel  : MC_ReadActualVelocity; (* fbVel is instance of MC_ReadActualVelocity *)

END_VAR
-----------------------------------------------------------------------------------------------
(* Function Block Call for reading the actual velocity *)
fbVel(Axis := myAxis, Enable := TRUE);
```

Important
The execution of a function block instance might take longer than one PLC cycle. For a proper working system a function block instance has to be called (Execute or Enable) at every program cycle until its termination is signalled by the outputs Done, Error or Abort. At every call the function block instance will continue at its actual internal state where it has stopped during the previous PLC program cycle. Breaking this rule causes system errors. Especially if the function block uses CAN communication services which might not be finished fast enough.

7.1.14 Function Block MC_ReadActualCurrent

The Function Block MC_ReadActualCurrent returns the actual current of an axis.

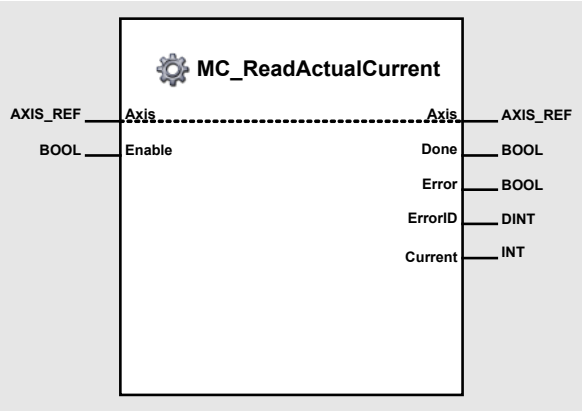


Figure 67: MC_ReadActualCurrent

Input/Output Variable

Name	Data Type	Elements	Element Type	Default Value	Value Range
Axis	AXIS_REF	AxisNo	USINT	0	0 to 31

The input/output variable Axis defines the addressed axis. With the “Network Configuration” tool the axis number of the internal and the external axis can be set freely within the allowed value range.

Input Variable

Name	Data Type	Default Value	Value Range	Unit
Enable	BOOL	FALSE	TRUE, FALSE	-

As long as the Enable input is TRUE (positive state), the value of the actual current is read continuously.

Output Variables

Name	Data Type	Default Value	Value Range	Unit
Done	BOOL	FALSE	FALSE, TRUE	-
Error	BOOL	FALSE	FALSE, TRUE	-
ErrorID	DINT	0	See chapter 10.2 Motion Control Functionblocks Error Codes	-
Current	INT	0	- 32768 [min(INT)] to + 32767 [max(INT)]	mA

A successful read operation is signalled by the output Done. The actual current can be read from the output Current. An error during execution of the function block is signalled at the output Error. Additionally an ErrorID allows to get more information about the error cause.

Function Block Call using the programming language ST

```
-----------------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis  : AXIS_REF := (AxisNo := 0);
fbCur   : MC_ReadActualCurrent; (* fbCur is instance of MC_ReadActualCurrent *)

END_VAR
-----------------------------------------------------------------------------------------------
(* Function Block Call for reading the actual current *)
fbCur(Axis := myAxis, Enable := TRUE);
```

Important

A declared function block instance has to be called every program cycle. Even if no functionality is executed at the The execution of a function block instance might take longer than one PLC cycle. For a proper working system a function block instance has to be called (Execute or Enable) at every program cycle until its termination is signalled by the outputs Done, Error or Abort. At every call the function block instance will continue at its actual internal state where it has stopped during the previous PLC program cycle. Breaking this rule causes system errors. Especially if the function block uses CAN communication services which might not be finished fast enough.

7.1.15 Function Block MC_ReadAxisError

The Function Block MC_ReadAxisError returns the first entry in the error history of the EPOS device.

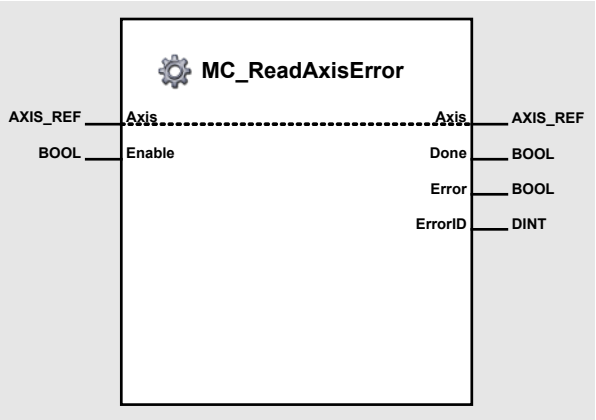


Figure 68: MC_ReadAxisError

Input Variable

Name	Data Type	Default Value	Value Range	Unit
Enable	BOOL	FALSE	TRUE, FALSE	-

As long as the Enable input is TRUE (positive state), the value of the first entry in the error history of the EPOS is read continuously.

Output Variables

Name	Data Type	Default Value	Value Range	Unit
Done	BOOL	FALSE	FALSE, TRUE	-
Error	BOOL	FALSE	Always FALSE	-
ErrorID	DINT	0	EPOS device Error Code [7]	-

A successful read operation is signalled by the output Done. An error during execution of the function block is signalled at the output Error. In case of a successful operation (Error = FALSE) the output ErrorID contains the axis error (see [7]) .

7.2 Maxon Utility Function Blocks

7.2.1 Function Block MU_GetAllDigitalInputs

The Function Block MU_GetAllDigitalInputs returns the state of all digital inputs.

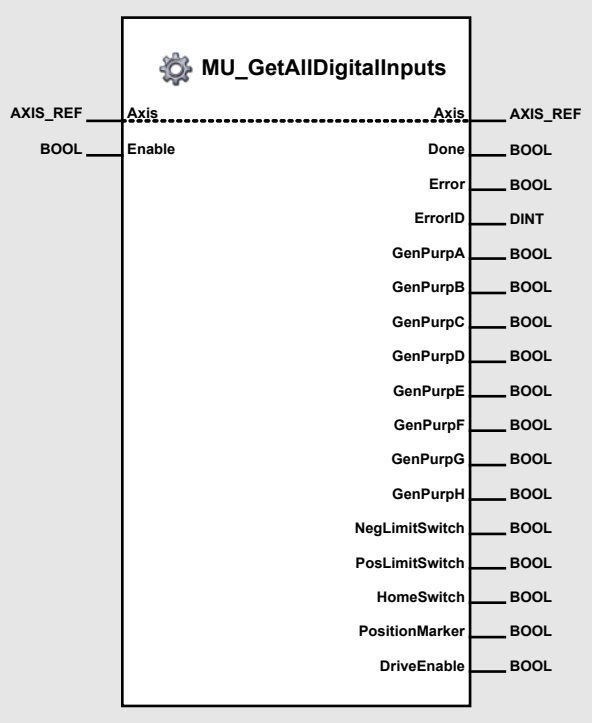


Figure 69: MU_GetAllDigitalInputs

Input/Output Variable

Name	Data Type	Elements	Element Type	Default Value	Value Range
Axis	AXIS_REF	AxisNo	USINT	0	0 to 31

The input/output variable Axis defines the addressed axis. With the “Network Configuration” tool the axis number of the internal and the external axis can be set freely within the allowed value range.

Input Variable

Name	Data Type	Default Value	Value Range	Unit
Enable	BOOL	FALSE	TRUE, FALSE	-

As long as the Enable input is TRUE (positive state), the status of all digital inputs is read continuously.

Output Variables

Name	Data Type	Default Value	Value Range	Unit
Done	BOOL	FALSE	FALSE, TRUE	-
Error	BOOL	FALSE	FALSE, TRUE	-
ErrorID	DINT	0	See chapter 10.2 Motion Control Functionblocks Error Codes	-
GenPurpA	BOOL	FALSE	FALSE, TRUE	-
GenPurpB	BOOL	FALSE	FALSE, TRUE	-
GenPurpC	BOOL	FALSE	FALSE, TRUE	-
GenPurpD	BOOL	FALSE	FALSE, TRUE	-
GenPurpE	BOOL	FALSE	FALSE, TRUE	-
GenPurpF	BOOL	FALSE	FALSE, TRUE	-
GenPurpG	BOOL	FALSE	FALSE, TRUE	-
GenPurpH	BOOL	FALSE	FALSE, TRUE	-
NegLimitSwitch	BOOL	FALSE	FALSE, TRUE	-
PosLimitSwitch	BOOL	FALSE	FALSE, TRUE	-
HomeSwitch	BOOL	FALSE	FALSE, TRUE	-
PositionMarker	BOOL	FALSE	FALSE, TRUE	-
DriveEnable	BOOL	FALSE	FALSE, TRUE	-

A successful read operation is signalled by the output Done. An error during execution of the function block is signalled at the output Error. Additionally an ErrorID allows to get more information about the error cause.

Function Block Call using the programming language ST

```
-----------------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis                 : AXIS_REF := (AxisNo := 0);
fbGetAllDigitalInputs  : MU_GetAllDigitalInputs; (* fbGetAllDigitalInputs is instance of
                         MU_GetAllDigitalInputs *)

END_VAR
-----------------------------------------------------------------------------------------------
(* Function block call for reading the status of all digital inputs *)
fbGetAllDigitalInputs(Axis := myAxis, Enable := TRUE);
```

7.2.2 Function Block MU_GetDigitalInput

The Function Block MU_GetDigitalInput returns the state of a specific digital input.

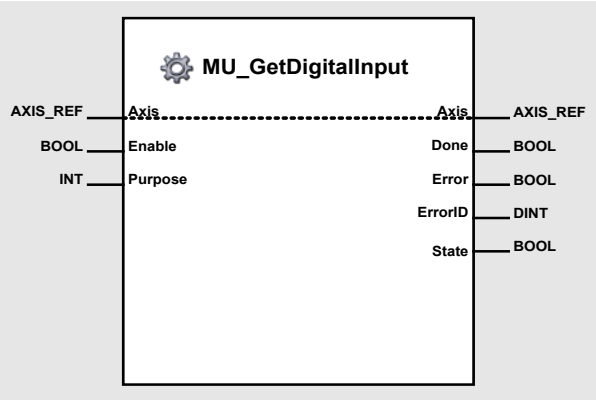


Figure 70: MU_GetDigitalInput

Input/Output Variable

Name	Data Type	Elements	Element Type	Default Value	Value Range
Axis	AXIS_REF	AxisNo	USINT	0	0 to 31

The input/output variable Axis defines the addressed axis. With the “Network Configuration” tool the axis number of the internal and the external axis can be set freely within the allowed value range.

Input Variable

Name	Data Type	Default Value	Value Range		Unit
Enable	BOOL	FALSE	TRUE, FALSE		-
Purpose	INT	0	NegLimitSwitch PosLimitSwitch HomeSwitch PositionMarker Enable GenPurpH GenPurpG GenPurpF GenPurpE GenPurpD GenPurpC GenPurpB GenPurpA	= 0, = 1, = 2, = 3, = 4, = 8, = 9, = 10, = 11, = 12, = 13, = 14, = 15	-

As long as the Enable input is TRUE (positive state), the status of a digital input is read continuously. The input variable Purpose defines the digital input to be read.

Output Variables

Name	Data Type	Default Value	Value Range	Unit
Done	BOOL	FALSE	FALSE, TRUE	-
Error	BOOL	FALSE	FALSE, TRUE	-
ErrorID	DINT	0	See chapter 10.2 Motion Control Functionblocks Error Codes	-
State	BOOL	FALSE	FALSE, TRUE	-

A successful read operation is signalled by the output Done. An error during execution of the function block is signalled at the output Error. Additionally an ErrorID allows to get more information about the error cause.

Function Block Call using the programming language ST

```
------------------------------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis                : AXIS_REF := (AxisNo := 0);
fbGetDigitalInput     : MU_GetDigitalInput; (* fbGetDigitalInput is instance of MU_GetDigitalInput *)

END_VAR
------------------------------------------------------------------------------------------------------------
(* Function Block Call for reading the status of home switch *)
fbGetDigitalInput(Axis := myAxis, Enable := TRUE, Purpose :=2);
```

7.2.3 Function Block MU_GetAnalogInput

The Function Block MU_GetAnalogInput returns the value of a specific analog input.

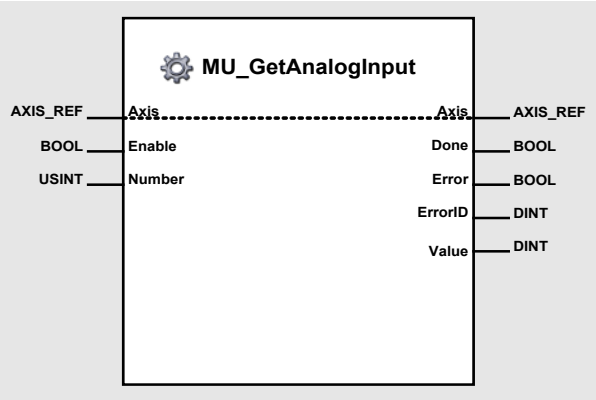


Figure 71: MU_GetAnalogInput

Input/Output Variable

Name	Data Type	Elements	Element Type	Default Value	Value Range
Axis	AXIS_REF	AxisNo	USINT	0	0 to 31

The input/output variable Axis defines the addressed axis. With the “Network Configuration” tool the axis number of the internal and the external axis can be set freely within the allowed value range.

Input Variable

Name	Data Type	Default Value	Value Range	Unit
Enable	BOOL	FALSE	TRUE, FALSE	-
Number	USINT	0	1, 2	-

As long as the Enable input is TRUE (positive state), the value of an analog input is read continuously. The input variable Number defines the analog input to be read.

Output Variables

Name	Data Type	Default Value	Value Range	Unit
Done	BOOL	FALSE	FALSE, TRUE	-
Error	BOOL	FALSE	FALSE, TRUE	-
ErrorID	DINT	0	See chapter 10.2 Motion Control Functionblocks Error Codes	-
Value	DINT	0	0 to 5’000	mV

A successful read operation is signalled by the output Done. The value can be read from the output Value. An error during execution of the function block is signalled at the output Error. Additionally an ErrorID allows to get more information about the error cause.

Function Block Call using the programming language ST

```
------------------------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis                : AXIS_REF := (AxisNo := 0);
fbGetAnalogInput      : MU_GetAnalogInput; (* fbGetAnalogInput is instance of MU_GetAnalogInput *)

END_VAR
------------------------------------------------------------------------------------------------------
(* Function Block Call for reading the value of the analog input 2 *)
fbGetAnalogInput(Axis := myAxis, Enable := TRUE, Number :=2);
```

7.2.4 Function Block MU_SetAllDigitalOutputs

The Function Block MU_SetAllDigitalOutputs modifies the value of all digital outputs.

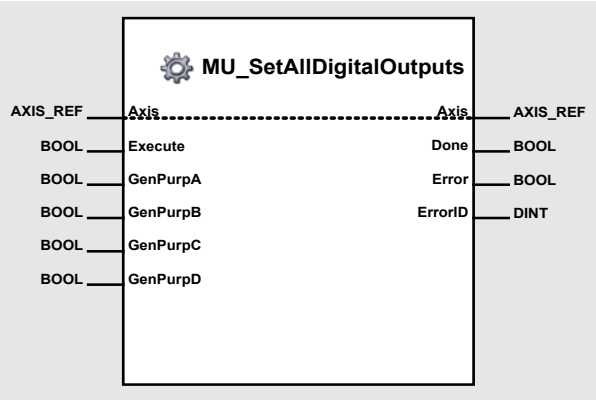


Figure 72: MU_SetAllDigitalOutputs

Input/Output Variable

Name	Data Type	Elements	Element Type	Default Value	Value Range
Axis	AXIS_REF	AxisNo	USINT	0	0 to 31

The input/output variable Axis defines the addressed axis. With the “Network Configuration” tool the axis number of the internal and the external axis can be set freely within the allowed value range.

Input Variable

Name	Data Type	Default Value	Value Range	Unit
Execute	BOOL	FALSE	TRUE, FALSE	-
GenPurpA	BOOL	FALSE	TRUE, FALSE	-
GenPurpB	BOOL	FALSE	TRUE, FALSE	-
GenPurpC	BOOL	FALSE	TRUE, FALSE	-
GenPurpD	BOOL	FALSE	TRUE, FALSE	-

A positive edge at the input Execute triggers a write operation of all digital outputs. The inputs GenPurpA, GenPurpB, GenPurpC and GenPurpD contain the new output value.

Output Variables

Name	Data Type	Default Value	Value Range	Unit
Done	BOOL	FALSE	FALSE, TRUE	-
Error	BOOL	FALSE	FALSE, TRUE	-
ErrorID	DINT	0	See chapter 10.2 Motion Control Functionblocks Error Codes	-

A successful read operation is signalled by the output Done. An error during execution of the function block is signalled at the output Error. Additionally an ErrorID allows to get more information about the error cause.

Function Block Call using the programming language ST

```
-----------------------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis                : AXIS_REF := (AxisNo := 0);
fbSetAllDigitalOutputs : MU_SetAllDigitalOutputs; (* fbSetAllDigitalOutputs is instance of
                         MU_SetAllDigitalOutputs *)

END_VAR
-----------------------------------------------------------------------------------------------------
(* Function Block Call for setting the value of all digital outputs to TRUE *)
fbGetAnalogInput(Axis := myAxis, Execute := TRUE, GenPurpA := TRUE, GenPurpB := TRUE, GenPurpC := TRUE,
GenPurpD := TRUE);
```

7.2.5 Function Block MU_GetDeviceErrorCount

The Function Block MU_GetDeviceErrorCount returns the number of actual errors.

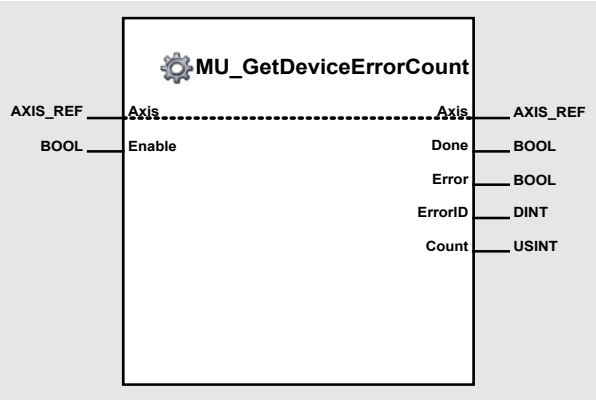


Figure 73: MU_GetDeviceErrorCount

Input/Output Variable

Name	Data Type	Elements	Element Type	Default Value	Value Range
Axis	AXIS_REF	AxisNo	USINT	0	0 to 31

The input/output variable Axis defines the addressed axis. With the “Network Configuration” tool the axis number of the internal and the external axis can be set freely within the allowed value range.

Input Variable

Name	Data Type	Default Value	Value Range	Unit
Enable	BOOL	FALSE	TRUE, FALSE	-

As long as the Enable input is TRUE (positive state), the number of existing errors is read continuously.

Output Variables

Name	Data Type	Default Value	Value Range	Unit
Done	BOOL	FALSE	FALSE, TRUE	-
Error	BOOL	FALSE	FALSE, TRUE	-
ErrorID	DINT	0	See chapter 10.2 Motion Control Functionblocks Error Codes	-
Count	USINT	0	0 to 255	-

A successful read operation is signalled by the output Done. The actual number of existing errors can be read from the output Count. An error during execution of the function block is signalled at the output Error. Additionally an ErrorID allows to get more information about the error cause.

Function Block Call using the programming language ST

```
-----------------------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis                : AXIS_REF := (AxisNo := 0);
fbGetDeviceErrorCount : MU_GetDeviceErrorCount; (* fbGetDeviceErrorCount is instance of
                        MU_GetDeviceErrorCount *)

END_VAR
-----------------------------------------------------------------------------------------------------
(* Function Block Call for reading the number of existing errors *)
fbGetDeviceErrorCount(Axis := myAxis, Enable := TRUE);
```

7.2.6 Function Block MU_GetDeviceError

The Function Block MU_GetDeviceError returns the error code of a specific entry in the error history.

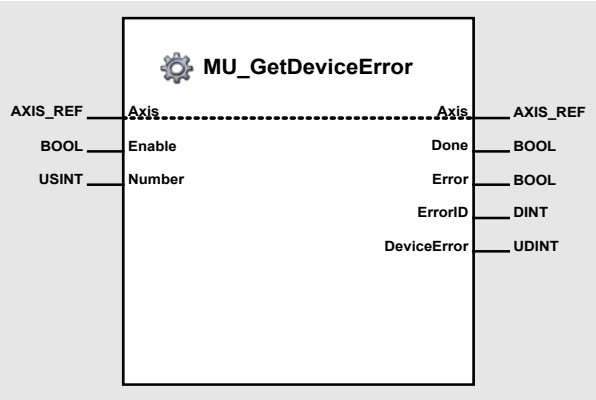


Figure 74: MU_GetDeviceError

Input/Output Variable

Name	Data Type	Elements	Element Type	Default Value	Value Range
Axis	AXIS_REF	AxisNo	USINT	0	0 to 31

The input/output variable Axis defines the addressed axis. With the “Network Configuration” tool the axis number of the internal and the external axis can be set freely within the allowed value range.

Input Variable

Name	Data Type	Default Value	Value Range	Unit
Enable	BOOL	FALSE	TRUE, FALSE	-
Number	USINT	1	1 to Count See chapter 7.2.5 Function Block MU_GetDeviceErrorCount	-

As long as the Enable input is TRUE (positive state), the error code of a specific entry in the error history is read continuously. The input Number defines the entry in the error history to be read.

Output Variables

Name	Data Type	Default Value	Value Range	Unit
Done	BOOL	FALSE	FALSE, TRUE	-
Error	BOOL	FALSE	FALSE, TRUE	-
ErrorID	DINT	0	See chapter 10.2 Motion Control Functionblocks Error Codes	-
DeviceError	UDINT	0	0 to 4 294 967 265	-

A successful read operation is signalled by the output Done. The error code can be read from the output DeviceError. An error during execution of the function block is signalled at the output Error. Additionally an ErrorID allows to get more information about the error cause.

Function Block Call using the programming language ST

```
-----------------------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis                : AXIS_REF := (AxisNo := 0);
fbGetDeviceError      : MU_GetDeviceError; (* fbGetDeviceError is instance of MU_GetDeviceError *)

END_VAR
-----------------------------------------------------------------------------------------------------
(* Function Block Call for reading the error code of the second entry in the error history *)
fbGetDeviceErrorCount(Axis := myAxis, Enable := TRUE, Number := 2);
```

7.2.7 Function Block MU_GetObject

The Function Block MU_GetObject returns the value of an EPOS object.

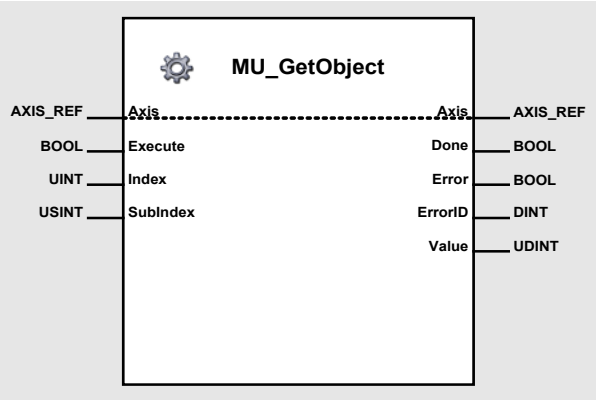


Figure 75: MU_GetObject

Input/Output Variable

Name	Data Type	Elements	Element Type	Default Value	Value Range
Axis	AXIS_REF	AxisNo	USINT	0	0 to 31

The input/output variable Axis defines the addressed axis. With the “Network Configuration” tool the axis number of the internal and the external axis can be set freely within the allowed value range.

Input Variable

Name	Data Type	Default Value	Value Range	Unit
Execute	BOOL	FALSE	TRUE, FALSE	-
Index	UINT	0	0 to 65 535	-
SubIndex	USINT	0	0 to 255	

A positive edge at the input Execute triggers a read operation of a specific EPOS object. The inputs Index and SubIndex define the object to be read.

Output Variables

Name	Data Type	Default Value	Value Range	Unit
Done	BOOL	FALSE	FALSE, TRUE	-
Error	BOOL	FALSE	FALSE, TRUE	-
ErrorID	DINT	0	See chapter 10.2 Motion Control Functionblocks Error Codes	-
Value	UDINT	0	0 to 4 294 967 265	-

A successful read operation is signalled by the output Done. The value of the object can be read from the output Value. An error during execution of the function block is signalled at the output Error. Additionally an ErrorID allows to get more information about the error cause.

Function Block Call using the programming language ST

```
-----------------------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis                : AXIS_REF := (AxisNo := 0);
fbGetObject           : MU_GetObject; (* fbGetObject is instance of MU_GetObject *)

END_VAR
-----------------------------------------------------------------------------------------------------
(* Function Block Call for reading the software number of the attached EPOS (object: 0x2003-01 *)
fbGetObject(Axis := myAxis, Execute := TRUE, Index := 16#2003, SubIndex := 16#01);
```

7.2.8 Function Block MU_SetObject

The Function Block MU_SetObject modifies the value of an EPOS object.

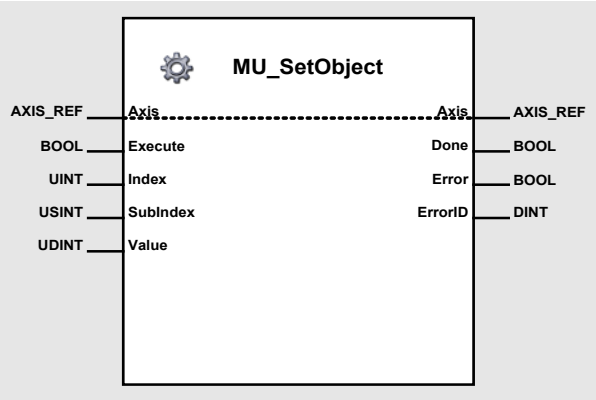


Figure 76: MU_SetObject

Input/Output Variable

Name	Data Type	Elements	Element Type	Default Value	Value Range
Axis	AXIS_REF	AxisNo	USINT	0	0 to 31

The input/output variable Axis defines the addressed axis. With the “Network Configuration” tool the axis number of the internal and the external axis can be set freely within the allowed value range.

Input Variable

Name	Data Type	Default Value	Value Range	Unit
Execute	BOOL	FALSE	TRUE, FALSE	-
Index	UINT	0	0 to 65 535	-
SubIndex	USINT	0	0 to 255	
Value	UDINT	0	0 to 4 294 967 265	-

A positive edge at the input Execute triggers a write operation of a specific EPOS object. The inputs Index and SubIndex define the object to be modified. The input Value contains the new object value.

Output Variables

Name	Data Type	Default Value	Value Range	Unit
Done	BOOL	FALSE	FALSE, TRUE	-
Error	BOOL	FALSE	FALSE, TRUE	-
ErrorID	DINT	0	See chapter 10.2 Motion Control Functionblocks Error Codes	-

A successful write operation is signalled by the output Done. An error during execution of the function block is signalled at the output Error. Additionally an ErrorID allows to get more information about the error cause.

Function Block Call using the programming language ST

```
-----------------------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis                : AXIS_REF := (AxisNo := 0);
fbSetObject           : MU_SetObject; (* fbSetObject is instance of MU_SetObject *)

END_VAR
-----------------------------------------------------------------------------------------------------
(* Function Block Call for writing the encoder pulse number of the attached EPOS (object: 0x2210-01 *)
fbSetObject(Axis := myAxis, Execute := TRUE, Index := 16#2210, SubIndex := 16#01);
```

7.2.9 Function Block MU_GetHomingParameter

The Function Block MU_GetHomingParameter returns the values of the EPOS homing objects.

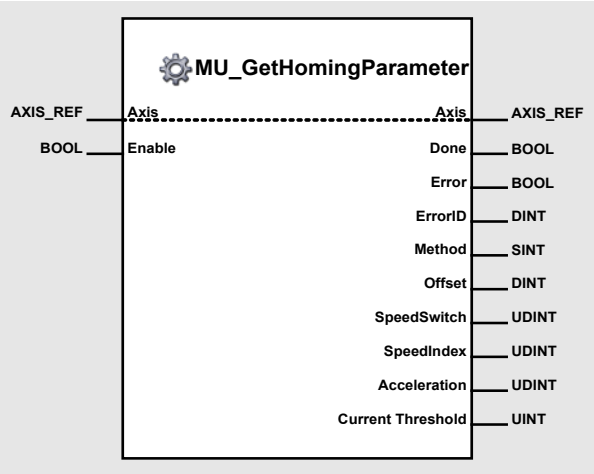


Figure 77: MU_GetHomingParameter

Input/Output Variable

Name	Data Type	Elements	Element Type	Default Value	Value Range
Axis	AXIS_REF	AxisNo	USINT	0	0 to 31

The input/output variable Axis defines the addressed axis. With the “Network Configuration” tool the axis number of the internal and the external axis can be set freely within the allowed value range.

Input Variable

Name	Data Type	Default Value	Value Range	Unit
Enable	BOOL	FALSE	TRUE, FALSE	-

As long as the Enable input is TRUE (positive state), the values of the EPOS homing objects are read continuously.

Output Variables

Name	Data Type	Default Value	Value Range	Unit
Done	BOOL	FALSE	FALSE, TRUE	-
Error	BOOL	FALSE	FALSE, TRUE	-
ErrorID	DINT	0	See chapter 10.2 Motion Control Functionblocks Error Codes	-
Method	SINT	7	cNegLimitSwitchIndex = 1, cPosLimitSwitchIndex = 2, cHomeSwitchPosSpeedIndex = 7, cHomeSwitchNegSpeedIndex = 11, cNegLimitSwitch = 17, cPosLimitSwitch = 18, cHomeSwitchPosSpeed = 23, cHomeSwitchNegSpeed = 27, cIndexNegSpeed = 33, cIndexPosSpeed = 34, cActualPosition = 35, cCurThreshPosSpeedIndex = -1, cCurThreshNegSpeedIndex = -2, cCurThreshPosSpeed = -3, cCurThreshNegSpeed = -4	-
Offset	DINT	0	-2147483648 to 2147483647	qc
SpeedSwitch	UDINT	100	0 to 4294967295	rpm
SpeedIndex	UDINT	10	0 to 4294967295	rpm
Acceleration	UDINT	1000	0 to 4294967295	rpm / s
CurrentThreshold	UINT	500	0 to depend on hardware	mA

A successful read operation is signalled by the output Done. An error during execution of the function block is signalled at the output Error. Additionally an ErrorID allows to get more information about the error cause. The values of the objects can be read from the outputs Method, Offset, SpeedSwitch, SpeedIndex, Acceleration and CurrentThreshold.

Function Block Call using the programming language ST

```
-----------------------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis                 : AXIS_REF := (AxisNo := 0);
fbGetHomingParameter   : MU_GetHomingParameter; (* fbGetHomingParameter is instance of
                         MU_GetHomingParameter *)

END_VAR
-----------------------------------------------------------------------------------------------------
(* Function Block Call for reading the homing parameter *)
fbGetHomingParameter(Axis := myAxis, Enable := TRUE);
```

7.2.10 Function Block MU_SetHomingParameter

The Function Block MU_SetHomingParameter modifies the values of the EPOS homing objects.

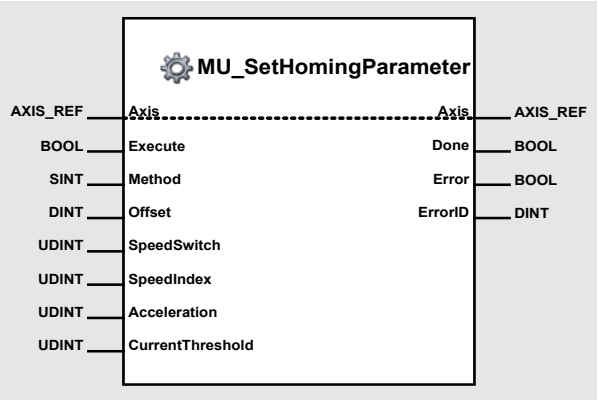


Figure 78: MU_SetHomingParameter

Input/Output Variable

Name	Data Type	Elements	Element Type	Default Value	Value Range
Axis	AXIS_REF	AxisNo	USINT	0	0 to 31

The input/output variable Axis defines the addressed axis. With the “Network Configuration” tool the axis number of the internal and the external axis can be set freely within the allowed value range.

Input Variable

Name	Data Type	Default Value	Value Range	Unit
Execute	BOOL	FALSE	TRUE, FALSE	-
Method	SINT	7	cNegLimitSwitchIndex = 1, cPosLimitSwitchIndex = 2, cHomeSwitchPosSpeedIndex = 7, cHomeSwitchNegSpeedIndex = 11, cNegLimitSwitch = 17, cPosLimitSwitch = 18, cHomeSwitchPosSpeed = 23, cHomeSwitchNegSpeed = 27, cIndexNegSpeed = 33, cIndexPosSpeed = 34, cActualPosition = 35, cCurThreshPosSpeedIndex = -1, cCurThreshNegSpeedIndex = -2, cCurThreshPosSpeed = -3, cCurThreshNegSpeed = -4	-
Offset	DINT	0	-2147483648 to 2147483647	qc
SpeedSwitch	UDINT	100	0 to 4294967295	rpm
SpeedIndex	UDINT	10	0 to 4294967295	rpm
Acceleration	UDINT	1000	0 to 4294967295	rpm / s
CurrentThreshold	UINT	500	0 to depend on hardware	mA

A positive edge at the input Execute triggers a write operation of the EPOS homing objects. The inputs Method, Offset, SpeedSwitch, SpeedIndex, Acceleration and CurrentThreshold contain the value of the parameters to be written.

Output Variables

Name	Data Type	Default Value	Value Range	Unit
Done	BOOL	FALSE	FALSE, TRUE	-
Error	BOOL	FALSE	FALSE, TRUE	-
ErrorID	DINT	0	See chapter 10.2 Motion Control Functionblocks Error Codes	-

A successful read operation is signalled by the output Done. An error during execution of the function block is signalled at the output Error. Additionally an ErrorID allows to get more information about the error cause.

Function Block Call using the programming language ST

```
-----------------------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis                 : AXIS_REF := (AxisNo := 0);
fbSetHomingParameter   : MU_SetHomingParameter; (* fbSetHomingParameter is instance of
                         MU_SetHomingParameter *)

END_VAR
-----------------------------------------------------------------------------------------------------
(* Function Block Call for writing the homing parameter *)
fbSetHomingParameter(Axis := myAxis, Execute := TRUE, Method :=11, Offset:= 200, SpeedSwitch := 150,
SpeedIndex := 20, Acceleration := 2000, CurrentThreshold := 500);
```

7.2.11 Function Block MU_Selection

The Function Block MU_Selection selects between two values.

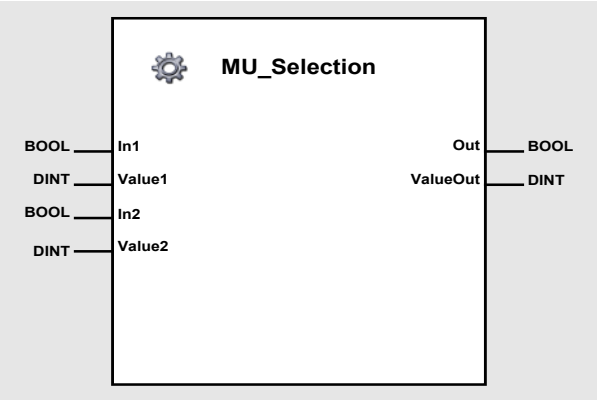


Figure 79: MU_Selection

Input Variables

Name	Data Type	Default Value	Value Range	Unit
In1	BOOL	FALSE	TRUE, FALSE	-
Value1	DINT	0	-2147483648 to 2147483647	-
In2	BOOL	FALSE	TRUE, FALSE	-
Value2	DINT	0	-2147483648 to 2147483647	-

With the input variables In1 and In2, one of the input variables Value1 or Value2 can be selected. If In1 and In2 are TRUE, In1 is prioritized.

Output Variables

Name	Data Type	Default Value	Value Range	Unit
Out	BOOL	FALSE	FALSE, TRUE	-
ValueOut	DINT	0	-2147483648 to 2147483647	-

The output variable Out indicates a valid value of the output variable ValueOut.

Function Block Call using the programming language ST

```
-----------------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
fbSelection : MU_Selection; (* fbSelection is instance of MU_Selection *)

END_VAR
-----------------------------------------------------------------------------------------------
(* Function Block call for selecting Value1 input *)
fbSelection(In1 := TRUE, Value1 := 2000, In2 := FALSE, Value2 := 1000);
```

7.2.12 Function Block MU_GetBitState

The Function Block MU_GetBitState extracts the state of a specific bit.

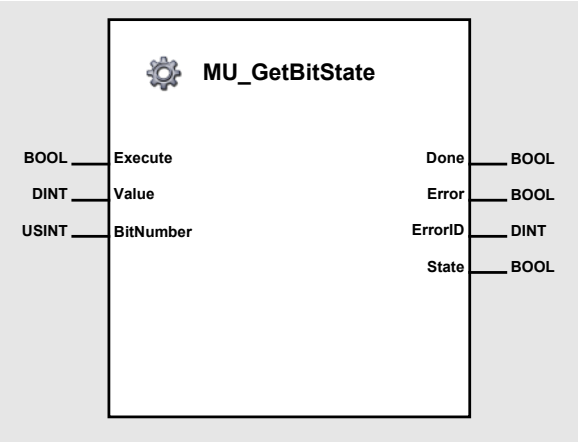


Figure 80: MU_GetBitState

Input Variables

Name	Data Type	Default Value	Value Range	Unit
Execute	BOOL	FALSE	TRUE, FALSE	-
Value	DINT	0	-2147483648 to 2147483647	-
BitNumber	USINT	0	0 to 255	-

A positive edge at the input Execute triggers a read operation of the state of a specific bit within the input variable Value. The input variable BitNumber defines the bit to be read

Output Variables

Name	Data Type	Default Value	Value Range	Unit
Done	BOOL	FALSE	FALSE, TRUE	-
Error	BOOL	FALSE	FALSE, TRUE	-
ErrorID	DINT	0	See chapter 10.2 Motion Control Functionblocks Error Codes	-
State	BOOL	FALSE	FALSE, TRUE	-

A successful operation is signalled by the output Done. An error during execution of the function block is signalled at the output Error. Additionally an ErrorID allows to get more information about the error cause. The state of the bit can be read from the output State.

Function Block Call using the programming language ST

```
-----------------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
fbGetBitState : MU_GetBitState; (* fbGetBitState is instance of MU_GetBitState *)

END_VAR
-----------------------------------------------------------------------------------------------
(* Function Block call for reading state of bit 2 of Value*)
fbGetBitState(Execute := TRUE, Value := 2#10010000, BitNumber := 2);

(* Return value of function block: State = 0*)
```

7.2.13 Function Block MU_SetBitState

The Function Block MU_SetBitState modifies the state of a specific bit within a given value.

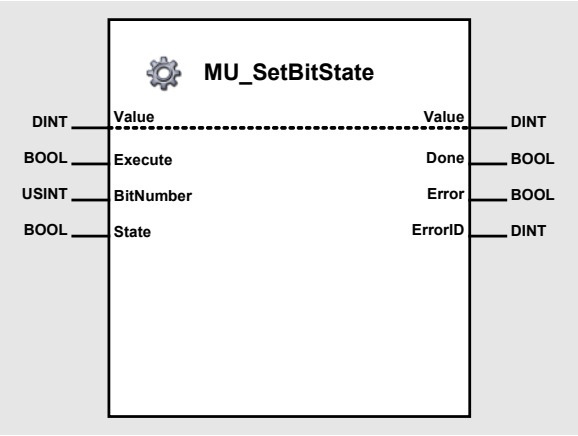


Figure 81: MU_SetBitState

Input/Output Variable

Name	Data Type	Default Value	Value Range	Unit
Value	DINT	0	-2147483648 to 2147483647	-

The input/output variable Value contains the value within a specific bit will be modified.

Input Variables

Name	Data Type	Default Value	Value Range	Unit
Execute	BOOL	FALSE	TRUE, FALSE	-
BitNumber	USINT	0	0 to 255	-
State	BOOL	FALSE	FALSE, TRUE	-

A positive edge at the input Execute triggers a write operation of the state of a specific bit within the input variable Value. The input variable BitNumber defines the bit to be written with the value in the input variable State.

Output Variables

Name	Data Type	Default Value	Value Range	Unit
Done	BOOL	FALSE	FALSE, TRUE	-
Error	BOOL	FALSE	FALSE, TRUE	-
ErrorID	DINT	0	See chapter 10.2 Motion Control Functionblocks Error Codes	-

A successful operation is signalled by the output Done. An error during execution of the function block is signalled at the output Error. Additionally an ErrorID allows to get more information about the error cause.

Function Block Call using the programming language ST

```
-----------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
fbSetBitState : MU_SetBitState; (* fbSetBitState is instance of MU_SetBitState *)

END_VAR
-----------------------------------------------------------------------------------------
(* Function Block call for writing bit 2 of Value with state = TRUE*)
fbSetBitState(Execute := TRUE, Value := 2#10010000, BitNumber := 2, State := TRUE);

(* Content of variable Value before function block call:    2#10010000*)
(* Content of variable Value after function block call:     2#10010100*)
```

7.3 CANopen DS-301 Function Blocks

7.3.1 Function Block CAN_Nmt

With the Function Block CAN_Nmt the network management state of a CANopen device can be changed.

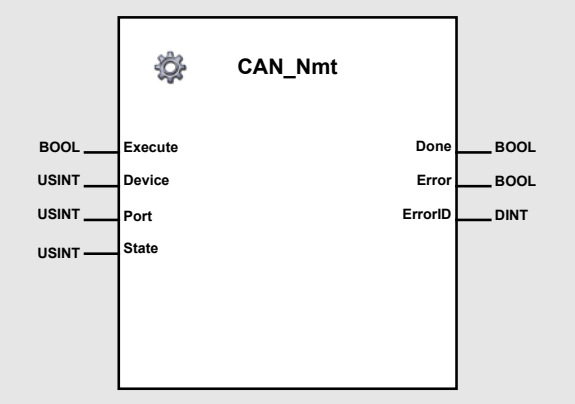


Figure 82: CAN_Nmt

Input Variables

Name	Data Type	Default Value	Value Range	Unit
Execute	BOOL	FALSE	TRUE, FALSE	-
Device	USINT	0	0 to 127	-
Port	USINT	0	0 internal port 1 external port	-
State	USINT	0	1 = Start Remote Node, 2 = Stop Remote Node, 128 = Enter Pre-Operational, 129 = Reset Node, 130 = Reset Communication	-

A positive edge of the input Execute is triggering the NMT service operation. The network management state of the defined device is changed. The input Device corresponds to the CAN Node-ID. A Device value of 0 changes the NMT state of all devices in the network selected by the input port. The input Port distinguishes between internal and external CAN network.
The NMT state machine is define by CANopen and is explained in the CANopen specification.

Output Variables

Name	Data Type	Default Value	Value Range	Unit
Done	BOOL	FALSE	FALSE, TRUE	-
Error	BOOL	FALSE	FALSE, TRUE	-
ErrorID	DINT	0	See chapter 10.2 Motion Control Functionblocks Error Codes	-

A successful operation is signalled by the output Done. An error during execution of the function block is signalled at the output Error. Additionally an ErrorID allows to get more information about the error cause.

Function Block Call using the programming language ST

```
-----------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
fbNmt : CAN_Nmt; (* fbNmt is instance of CAN_Nmt *)

END_VAR
-----------------------------------------------------------------------------------------
(* Function Block call for starting all nodes *)
fbNmt(Execute := TRUE, Device := 0, Port := 0, State := 1);
```

7.3.2 Function Block CAN_SdoRead

With the Function Block CAN_SdoRead a CANopen object can be read using the SDO protocol.

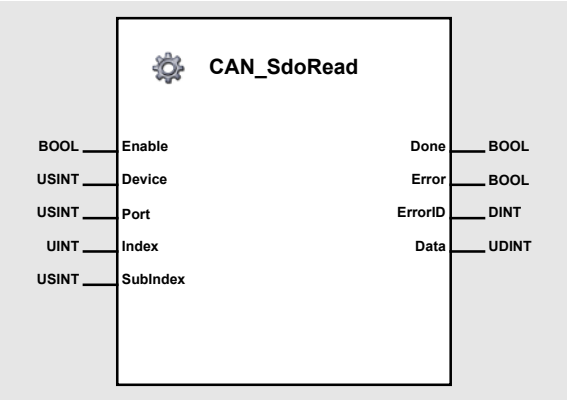


Figure 83: CAN_SdoRead

Input Variables

Name	Data Type	Default Value	Value Range	Unit
Enable	BOOL	FALSE	TRUE, FALSE	-
Device	USINT	0	0 to 127	-
Port	USINT	0	0 internal port 1 external port	-
Index	UINT	0	UINT	-
Sub-index	USINT	0	USINT	-

As long as the Enable input is TRUE (positive state), the value of a specified CANopen object is read continuously. The object is specified by the inputs Index and Sub-index. The input Device corresponds to the CAN Node-ID. The input Port distinguishes between internal and external CAN network.

Output Variables

Name	Data Type	Default Value	Value Range	Unit
Done	BOOL	FALSE	FALSE, TRUE	-
Error	BOOL	FALSE	FALSE, TRUE	-
ErrorID	DINT	0	See chapter 10.2 Motion Control Functionblocks Error Codes	-
Data	UDINT	0	0 to 4’294’967’295	-

A successful read operation is signalled by the output Done. An error during execution of the function block is signalled at the output Error. Additionally an ErrorID allows to get more information about the error cause.

Function Block Call using the programming language ST

```
-----------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
fbSdoRead : CAN_SdoRead; (* fbSdoRead is instance of CAN_SdoRead *)

END_VAR
-----------------------------------------------------------------------------------------
(* Function Block call for reading the CANopen object 'DeviceType' *)
fbSdoRead (Enable := TRUE, Device := 0, Port := 0, Index := 16#1000, Sub-index := 16#00);
```

Important

The execution of a function block instance might take longer than one PLC cycle. For a proper working system a function block instance has to be called (Execute or Enable) at every program cycle until its termination is signalled by the outputs Done, Error or Abort. At every call the function block instance will continue at its actual internal state where it has stopped during the previous PLC program cycle. Breaking this rule causes system errors. Especially if the function block uses CAN communication services which might not be finished fast enough.

7.3.3 Function Block CAN_SdoWrite

With the Function Block CAN_SdoWrite a CANopen object can be written using the SDO protocol.

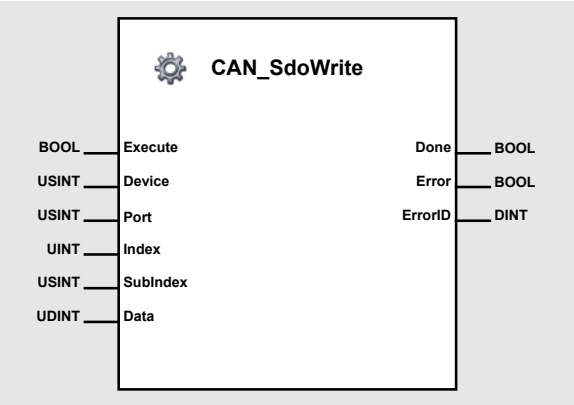


Figure 84: CAN_SdoWrite

Input Variables

Name	Data Type	Default Value	Value Range	Unit
Execute	BOOL	FALSE	TRUE, FALSE	-
Device	USINT	0	0 to 127	-
Port	USINT	0	0 internal port 1 external port	-
Index	UINT	0	UINT	-
Sub-index	USINT	0	USINT	-
Data	UDINT	0	UDINT	-

A positive edge at the input Execute triggers the write operation of a CANopen object. The object is specified by the inputs Index and Sub-index. The input Device corresponds to the CAN Node-ID. The input Port distinguishes between internal and external CAN network.

Output Variables

Name	Data Type	Default Value	Value Range	Unit
Done	BOOL	FALSE	FALSE, TRUE	-
Error	BOOL	FALSE	FALSE, TRUE	-
ErrorID	DINT	0	See chapter 10.2 Motion Control Functionblocks Error Codes	-

A successful write operation is signalled by the output Done. An error during execution of the function block is signalled at the output Error. Additionally an ErrorID allows to get more information about the error cause.

Function Block Call using the programming language ST

```
-----------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
fbSdoWrite : CAN_SdoWrite; (* fbSdoWrite is instance of CAN_SdoWrite *)

END_VAR
-----------------------------------------------------------------------------------------
(* Function Block call for writing the CANopen object 'GuardTime' *)
fbSdoWrite (Execute := TRUE, Device := 0, Port := 0, Index := 16#100C, Sub-index := 16#00, Data := 100);
```

Important

The execution of a function block instance might take longer than one PLC cycle. For a proper working system a function block instance has to be called (Execute or Enable) at every program cycle until its termination is signalled by the outputs Done, Error or Abort. At every call the function block instance will continue at its actual internal state where it has stopped during the previous PLC program cycle. Breaking this rule causes system errors. Especially if the function block uses CAN communication services which might not be finished fast enough.

8 Markers

Markers are typically used to build intermediate results. Those will be buffered in the PLC and do not have direct influence to the outputs. With markers extensive operations can be simplified essentially. Further, they act as transmitter between different modules.
The EPOS P is using specific marker areas for error and warning information's.

8.1 User Marker Area

The 'User Marker Area' length is 25 entries (32-bit values). It Is possible to write or read the 'User Marker Area'. To Access a marker variable IEC-61131 direct addressing method is used.

IEC-61131 declaration example with UDINT variables:

UserMarkerVariable0	AT	%MD0.0 : UDINT;
UserMarkerVariable1	AT	%MD4.0 : UDINT;
UserMarkerVariable2	AT	%MD8.0 : UDINT;
UserMarkerVariable3	AT	%MD12.0 :UDINT;
UserMarkerVariable4	AT	%MD16.0 : UDINT;
UserMarkerVariable5	AT	%MD20.0 : UDINT;
UserMarkerVariable6	AT	%MD24.0 : UDINT;
UserMarkerVariable7	AT	%MD28.0 : UDINT;
UserMarkerVariable8	AT	%MD32.0 : UDINT;
UserMarkerVariable9	AT	%MD36.0 : UDINT;
UserMarkerVariable10	AT	%MD40.0 : UDINT;
UserMarkerVariable11	AT	%MD44.0 : UDINT;
UserMarkerVariable12	AT	%MD48.0 : UDINT;
UserMarkerVariable13	AT	%MD52.0 : UDINT;
UserMarkerVariable14	AT	%MD56.0 : UDINT;
UserMarkerVariable15	AT	%MD60.0 : UDINT;
UserMarkerVariable16	AT	%MD64.0 : UDINT;
UserMarkerVariable17	AT	%MD68.0 : UDINT;
UserMarkerVariable18	AT	%MD72.0 : UDINT;
UserMarkerVariable19	AT	%MD76.0 : UDINT;
UserMarkerVariable20	AT	%MD80.0 : UDINT;
UserMarkerVariable21	AT	%MD84.0 : UDINT;
UserMarkerVariable22	AT	%MD88.0 : UDINT;
UserMarkerVariable23	AT	%MD92.0 : UDINT;
UserMarkerVariable24	AT	%MD96.0 : UDINT;

8.2 Marker Global Status Register

The 'Global Status Register' marker length is 32-bit. It holds the EPOS P global status register and is the very same register like the EPOS P CANopen object 0x1002. Bit 0 to Bit 7 represents an overview of the CANopen slave error register's. If one of the connected CANopen slave reports an error register flag, the according bit is set. For the meaning of CANopen error register please refer to the object description of the connected CANopen slave (object error register with index 0x1001 and subindex 0).

Bit	Description
0	One of the connected slaves is signalling a generic error bit in error register
1	One of the connected slaves is signalling a current error bit in error register
2	One of the connected slaves is signalling a voltage error bit in error register
3	One of the connected slaves is signalling a temperature error bit in error register
4	One of the connected slaves is signalling a communication error bit in error register
5	One of the connected slaves is signalling a device profile specific error bit in error register
6	Reserved
7	One of the connected slaves is signalling a manufacturer specific error bit in error register
8 - 15	Copy of error register
16	Master generic warning
17 - 19	Not used
20	Master communication warning
21 - 22	Not used
23	Master manufacturer specific warning
24 - 31	Not used

Table 1: Global Status Register Marker

IEC-61131 declaration example with BOOL variables:

ERR_mEposGenericError	AT	%M100.0 : BOOL;
ERR_mEposCurrentError	AT	%M100.1 : BOOL;
ERR_mEposVoltageError	AT	%M100.2 : BOOL;
ERR_mEposTemperatureError	AT	%M100.3 : BOOL;
ERR_mEposCommunicationError	AT	%M100.4 : BOOL;
ERR_mEposMotionError	AT	%M100.7 : BOOL;

8.3 Marker Global Axis Error Register

The 'Global Axis Error Register' length is 32-bit. It holds an overview of all connected axis. If one axis change to error state the according bit does indicate this.

Bit	Description
0	Axis **0** is in error state
1	Axis **1** is in error state
2	Axis **2** is in error state
n	Axis **n** is in error state
31	Axis **31** is in error state

Table 2: Global Axis Error Register Marker

IEC-61131 declaration example with BOOL variables:

ERR_mAxis0Error	AT	%M104.0 : BOOL;
ERR_mAxis1Error	AT	%M104.1 : BOOL;
ERR_mAxis2Error	AT	%M104.2 : BOOL;
ERR_mAxis3Error	AT	%M104.3 : BOOL;
ERR_mAxis4Error	AT	%M104.4 : BOOL;
ERR_mAxis5Error	AT	%M104.5 : BOOL;
ERR_mAxis6Error	AT	%M104.6 : BOOL;
ERR_mAxis7Error	AT	%M104.7 : BOOL;
ERR_mAxis8Error	AT	%M105.0 : BOOL;
ERR_mAxis9Error	AT	%M105.1 : BOOL;
ERR_mAxis10Error	AT	%M105.2 : BOOL;
ERR_mAxis11Error	AT	%M105.3 : BOOL;
ERR_mAxis12Error	AT	%M105.4 : BOOL;
ERR_mAxis13Error	AT	%M105.5 : BOOL;
ERR_mAxis14Error	AT	%M105.6 : BOOL;
ERR_mAxis15Error	AT	%M105.7 : BOOL;
ERR_mAxis16Error	AT	%M106.0 : BOOL;
ERR_mAxis17Error	AT	%M106.1 : BOOL;
ERR_mAxis18Error	AT	%M106.2 : BOOL;
ERR_mAxis19Error	AT	%M106.3 : BOOL;
ERR_mAxis20Error	AT	%M106.4 : BOOL;
ERR_mAxis21Error	AT	%M106.5 : BOOL;
ERR_mAxis22Error	AT	%M106.6 : BOOL;
ERR_mAxis23Error	AT	%M106.7 : BOOL;
ERR_mAxis24Error	AT	%M107.0 : BOOL;
ERR_mAxis25Error	AT	%M107.1 : BOOL;
ERR_mAxis26Error	AT	%M107.2 : BOOL;
ERR_mAxis27Error	AT	%M107.3 : BOOL;
ERR_mAxis28Error	AT	%M107.4 : BOOL;
ERR_mAxis29Error	AT	%M107.5 : BOOL;
ERR_mAxis30Error	AT	%M107.6 : BOOL;
ERR_mAxis31Error	AT	%M107.7 : BOOL;

8.4 Reserved Marker Area

The 'Reserved Marker Area' length is 23 entries (32-bit values). It is reserved for further use.

8.5 CANopen Slave Error Register Area

The 'CANopen Slave Error Register Area' length is 128 entries (8-bit values). It represents the CANopen error register of the connected slave. For the meaning of CANopen error register please refer to the object description of the connected CANopen slave (error register with index 0x1001 and subindex 0).

IEC-61131 declaration example with USINT variables:

ERR_mErrorRegisterInternalEPOS	AT	%MB200.0 : USINT;
ERR_mErrorRegisterCANopenSlave1	AT	%MB201.0 : USINT;
ERR_mErrorRegisterCANopenSlave2	AT	%MB202.0 : USINT;
...	AT	...
ERR_mErrorRegisterCANopenSlave127	AT	%MB327.0 : USINT;

IEC-61131 declaration example with BOOL variables for EPOS slaves (sample internal EPOS):

ERR_mInternalEposGenericError	AT	%M200.0 : BOOL;
ERR_mInternalEposCurrentError	AT	%M200.1 : BOOL;
ERR_mInternalEposVoltageError	AT	%M200.2 : BOOL;
ERR_mInternalEposTemperatureError	AT	%M200.3 : BOOL;
ERR_mInternalEposCommunicationError	AT	%M200.4 : BOOL;

9 Process Inputs and Outputs

Process inputs and outputs are used to read the incoming or write the outgoing CANopen PDO's. PDO communication is a powerful and easy way of communication by reading or writing direct addressed variables. Nevertheless before this communication method is ready to use a PDO configuration is necessary. Please refer to chapter Network Configuration. Use always the configuration tool for PDO configuration. For the ,option function blocks (e.g. MC_Power, ...) also use PDO communication.

9.1 Process Inputs

9.1.1 SINT Process Inputs

There are 16 SINT values in the 'Process Inputs Area' with the length of 8-bit. The variables holds the very same values like the related CANopen object dictionary entry with index 0xA000.

IEC-61131 declaration example with SINT variables:

InVar1	AT	%IB0.0 : SINT;	(*Object Index 0xA000, Subindex 1*)
InVar2	AT	%IB1.0 : SINT;	(*Object Index 0xA000, Subindex 2*)
InVar3	AT	%IB2.0 : SINT;	(*Object Index 0xA000, Subindex 3*)
InVar4	AT	%IB3.0 : SINT;	(*Object Index 0xA000, Subindex 4*)
InVar5	AT	%IB4.0 : SINT;	(*Object Index 0xA000, Subindex 5*)
InVar6	AT	%IB5.0 : SINT;	(*Object Index 0xA000, Subindex 6*)
InVar7	AT	%IB6.0 : SINT;	(*Object Index 0xA000, Subindex 7*)
InVar8	AT	%IB7.0 : SINT;	(*Object Index 0xA000, Subindex 8*)
InVar9	AT	%IB8.0 : SINT;	(*Object Index 0xA000, Subindex 9*)
InVar10	AT	%IB9.0 : SINT;	(*Object Index 0xA000, Subindex 10*)
InVar11	AT	%IB10.0 : SINT;	(*Object Index 0xA000, Subindex 11*)
InVar12	AT	%IB11.0 : SINT;	(*Object Index 0xA000, Subindex 12*)
InVar13	AT	%IB12.0 : SINT;	(*Object Index 0xA000, Subindex 13*)
InVar14	AT	%IB13.0 : SINT;	(*Object Index 0xA000, Subindex 14*)
InVar15	AT	%IB14.0 : SINT;	(*Object Index 0xA000, Subindex 15*)
InVar16	AT	%IB15.0 : SINT;	(*Object Index 0xA000, Subindex 16*)

9.1.2 USINT Process Inputs

There are 16 USINT values in the 'Process Inputs Area' with the length of 8-bit. The variables holds the very same values like the related CANopen object dictionary entry with index 0xA040.

IEC-61131 declaration example with USINT variables:

InVar1	AT	%IB16.0 : USINT;	(*Object Index 0xA040, Subindex 1*)
InVar2	AT	%IB17.0 : USINT;	(*Object Index 0xA040, Subindex 2*)
InVar3	AT	%IB18.0 : USINT;	(*Object Index 0xA040, Subindex 3*)
InVar4	AT	%IB19.0 : USINT;	(*Object Index 0xA040, Subindex 4*)
InVar5	AT	%IB20.0 : USINT;	(*Object Index 0xA040, Subindex 5*)
InVar6	AT	%IB21.0 : USINT;	(*Object Index 0xA040, Subindex 6*)
InVar7	AT	%IB22.0 : USINT;	(*Object Index 0xA040, Subindex 7*)
InVar8	AT	%IB23.0 : USINT;	(*Object Index 0xA040, Subindex 8*)
InVar9	AT	%IB24.0 : USINT;	(*Object Index 0xA040, Subindex 9*)
InVar10	AT	%IB25.0 : USINT;	(*Object Index 0xA040, Subindex 10*)
InVar11	AT	%IB26.0 : USINT;	(*Object Index 0xA040, Subindex 11*)
InVar12	AT	%IB27.0 : USINT;	(*Object Index 0xA040, Subindex 12*)
InVar13	AT	%IB28.0 : USINT;	(*Object Index 0xA040, Subindex 13*)
InVar14	AT	%IB29.0 : USINT;	(*Object Index 0xA040, Subindex 14*)
InVar15	AT	%IB30.0 : USINT;	(*Object Index 0xA040, Subindex 15*)
InVar16	AT	%IB31.0 : USINT;	(*Object Index 0xA040, Subindex 16*)

9.1.3 INT Process Inputs

There are 16 INT values in the 'Process Inputs Area' with the length of 16-bit. The variables holds the very same values like the related CANopen object dictionary entry with index 0xA0C0.

IEC-61131 declaration example with INT variables:

InVar1	AT	%IW32.0 : INT;	(*Object Index 0xA0C0, Subindex 1*)
InVar2	AT	%IW34.0 : INT;	(*Object Index 0xA0C0, Subindex 2*)
InVar3	AT	%IW36.0 : INT;	(*Object Index 0xA0C0, Subindex 3*)
InVar4	AT	%IW38.0 : INT;	(*Object Index 0xA0C0, Subindex 4*)
InVar5	AT	%IW40.0 : INT;	(*Object Index 0xA0C0, Subindex 5*)
InVar6	AT	%IW42.0 : INT;	(*Object Index 0xA0C0, Subindex 6*)
InVar7	AT	%IW44.0 : INT;	(*Object Index 0xA0C0, Subindex 7*)
InVar8	AT	%IW46.0 : INT;	(*Object Index 0xA0C0, Subindex 8*)
InVar9	AT	%IW48.0 : INT;	(*Object Index 0xA0C0, Subindex 9*)
InVar10	AT	%IW50.0 : INT;	(*Object Index 0xA0C0, Subindex 10*)
InVar11	AT	%IW52.0 : INT;	(*Object Index 0xA0C0, Subindex 11*)
InVar12	AT	%IW54.0 : INT;	(*Object Index 0xA0C0, Subindex 12*)
InVar13	AT	%IW56.0 : INT;	(*Object Index 0xA0C0, Subindex 13*)
InVar14	AT	%IW58.0 : INT;	(*Object Index 0xA0C0, Subindex 14*)
InVar15	AT	%IW60.0 : INT;	(*Object Index 0xA0C0, Subindex 15*)
InVar16	AT	%IW62.0 : INT;	(*Object Index 0xA0C0, Subindex 16*)

9.1.4 UINT Process Inputs

There are 16 UINT values in the 'Process Inputs Area' with the length of 16-bit. The variables holds the very same values like the related CANopen object dictionary entry with index 0xA100.

IEC-61131 declaration example with UINT variables:

InVar1	AT	%IW64.0 : UINT;	(*Object Index 0xA100, Subindex 1*)
InVar2	AT	%IW66.0 : UINT;	(*Object Index 0xA100, Subindex 2*)
InVar3	AT	%IW68.0 : UINT;	(*Object Index 0xA100, Subindex 3*)
InVar4	AT	%IW70.0 : UINT;	(*Object Index 0xA100, Subindex 4*)
InVar5	AT	%IW72.0 : UINT;	(*Object Index 0xA100, Subindex 5*)
InVar6	AT	%IW74.0 : UINT;	(*Object Index 0xA100, Subindex 6*)
InVar7	AT	%IW76.0 : UINT;	(*Object Index 0xA100, Subindex 7*)
InVar8	AT	%IW78.0 : UINT;	(*Object Index 0xA100, Subindex 8*)
InVar9	AT	%IW80.0 : UINT;	(*Object Index 0xA100, Subindex 9*)
InVar10	AT	%IW82.0 : UINT;	(*Object Index 0xA100, Subindex 10*)
InVar11	AT	%IW84.0 : UINT;	(*Object Index 0xA100, Subindex 11*)
InVar12	AT	%IW86.0 : UINT;	(*Object Index 0xA100, Subindex 12*)
InVar13	AT	%IW88.0 : UINT;	(*Object Index 0xA100, Subindex 13*)
InVar14	AT	%IW90.0 : UINT;	(*Object Index 0xA100, Subindex 14*)
InVar15	AT	%IW92.0 : UINT;	(*Object Index 0xA100, Subindex 15*)
InVar16	AT	%IW94.0 : UINT;	(*Object Index 0xA100, Subindex 16*)

9.1.5 DINT Process Inputs

There are 16 DINT values in the 'Process Inputs Area' with the length of 32-bit. The variables holds the very same values like the related CANopen object dictionary entry with index 0xA1C0.

IEC-61131 declaration example with DINT variables:

InVar1	AT	%ID96.0 : DINT;	(*Object Index 0xA1C0, Subindex 1*)
InVar2	AT	%ID100.0 : DINT;	(*Object Index 0xA1C0, Subindex 2*)
InVar3	AT	%ID104.0 : DINT;	(*Object Index 0xA1C0, Subindex 3*)
InVar4	AT	%ID108.0 : DINT;	(*Object Index 0xA1C0, Subindex 4*)
InVar5	AT	%ID112.0 : DINT;	(*Object Index 0xA1C0, Subindex 5*)
InVar6	AT	%ID116.0 : DINT;	(*Object Index 0xA1C0, Subindex 6*)
InVar7	AT	%ID120.0 : DINT;	(*Object Index 0xA1C0, Subindex 7*)
InVar8	AT	%ID124.0 : DINT;	(*Object Index 0xA1C0, Subindex 8*)
InVar9	AT	%ID128.0 : DINT;	(*Object Index 0xA1C0, Subindex 9*)
InVar10	AT	%ID132.0 : DINT;	(*Object Index 0xA1C0, Subindex 10*)
InVar11	AT	%ID136.0 : DINT;	(*Object Index 0xA1C0, Subindex 11*)
InVar12	AT	%ID140.0 : DINT;	(*Object Index 0xA1C0, Subindex 12*)
InVar13	AT	%ID144.0 : DINT;	(*Object Index 0xA1C0, Subindex 13*)
InVar14	AT	%ID148.0 : DINT;	(*Object Index 0xA1C0, Subindex 14*)
InVar15	AT	%ID152.0 : DINT;	(*Object Index 0xA1C0, Subindex 15*)
InVar16	AT	%ID156.0 : DINT;	(*Object Index 0xA1C0, Subindex 16*)

9.1.6 UDINT Process Inputs

There are 16 UDINT values in the 'Process Inputs Area' with the length of 32-bit. The variables holds the very same values like the related CANopen object dictionary entry with index 0xA200.

IEC-61131 declaration example with UDINT variables:

InVar1	AT	%ID160.0 : UDINT;	(*Object Index 0xA200, Subindex 1*)
InVar2	AT	%ID164.0 : UDINT;	(*Object Index 0xA200, Subindex 2*)
InVar3	AT	%ID168.0 : UDINT;	(*Object Index 0xA200, Subindex 3*)
InVar4	AT	%ID172.0 : UDINT;	(*Object Index 0xA200, Subindex 4*)
InVar5	AT	%ID176.0 : UDINT;	(*Object Index 0xA200, Subindex 5*)
InVar6	AT	%ID180.0 : UDINT;	(*Object Index 0xA200, Subindex 6*)
InVar7	AT	%ID184.0 : UDINT;	(*Object Index 0xA200, Subindex 7*)
InVar8	AT	%ID188.0 : UDINT;	(*Object Index 0xA200, Subindex 8*)
InVar9	AT	%ID192.0 : UDINT;	(*Object Index 0xA200, Subindex 9*)
InVar10	AT	%ID196.0 : UDINT;	(*Object Index 0xA200, Subindex 10*)
InVar11	AT	%ID200.0 : UDINT;	(*Object Index 0xA200, Subindex 11*)
InVar12	AT	%ID204.0 : UDINT;	(*Object Index 0xA200, Subindex 12*)
InVar13	AT	%ID208.0 : UDINT;	(*Object Index 0xA200, Subindex 13*)
InVar14	AT	%ID212.0 : UDINT;	(*Object Index 0xA200, Subindex 14*)
InVar15	AT	%ID216.0 : UDINT;	(*Object Index 0xA200, Subindex 15*)
InVar16	AT	%ID220.0 : UDINT;	(*Object Index 0xA200, Subindex 16*)

9.2 Process Outputs

9.2.1 SINT Process Outputs

There are 16 SINT values in the 'Process Outputs Area' with the length of 8-bit. The variables holds the very same values like the related CANopen object dictionary entry with index 0xA480.

IEC-61131 declaration example with SINT variables:

OutVar1	AT	%QB0.0 : SINT;	(*Object Index 0xA480, Subindex 1*)
OutVar2	AT	%QB1.0 : SINT;	(*Object Index 0xA480, Subindex 2*)
OutVar3	AT	%QB2.0 : SINT;	(*Object Index 0xA480, Subindex 3*)
OutVar4	AT	%QB3.0 : SINT;	(*Object Index 0xA480, Subindex 4*)
OutVar5	AT	%QB4.0 : SINT;	(*Object Index 0xA480, Subindex 5*)
OutVar6	AT	%QB5.0 : SINT;	(*Object Index 0xA480, Subindex 6*)
OutVar7	AT	%QB6.0 : SINT;	(*Object Index 0xA480, Subindex 7*)
OutVar8	AT	%QB7.0 : SINT;	(*Object Index 0xA480, Subindex 8*)
OutVar9	AT	%QB8.0 : SINT;	(*Object Index 0xA480, Subindex 9*)
OutVar10	AT	%QB9.0 : SINT;	(*Object Index 0xA480, Subindex 10*)
OutVar11	AT	%QB10.0 : SINT;	(*Object Index 0xA480, Subindex 11*)
OutVar12	AT	%QB11.0 : SINT;	(*Object Index 0xA480, Subindex 12*)
OutVar13	AT	%QB12.0 : SINT;	(*Object Index 0xA480, Subindex 13*)
OutVar14	AT	%QB13.0 : SINT;	(*Object Index 0xA480, Subindex 14*)
OutVar15	AT	%QB14.0 : SINT;	(*Object Index 0xA480, Subindex 15*)
OutVar16	AT	%QB15.0 : SINT;	(*Object Index 0xA480, Subindex 16*)

9.2.2 USINT Process Outputs

There are 16 USINT values in the 'Process Outputs Area' with the length of 8-bit. The variables holds the very same values like the related CANopen object dictionary entry with index 0xA4C0.

IEC-61131 declaration example with USINT variables:

OutVar1	AT	%QB16.0 : USINT;	(*Object Index 0xA4C0, Subindex 1*)
OutVar2	AT	%QB17.0 : USINT;	(*Object Index 0xA4C0, Subindex 2*)
OutVar3	AT	%QB18.0 : USINT;	(*Object Index 0xA4C0, Subindex 3*)
OutVar4	AT	%QB19.0 : USINT;	(*Object Index 0xA4C0, Subindex 4*)
OutVar5	AT	%QB20.0 : USINT;	(*Object Index 0xA4C0, Subindex 5*)
OutVar6	AT	%QB21.0 : USINT;	(*Object Index 0xA4C0, Subindex 6*)
OutVar7	AT	%QB22.0 : USINT;	(*Object Index 0xA4C0, Subindex 7*)
OutVar8	AT	%QB23.0 : USINT;	(*Object Index 0xA4C0, Subindex 8*)
OutVar9	AT	%QB24.0 : USINT;	(*Object Index 0xA4C0, Subindex 9*)
OutVar10	AT	%QB25.0 : USINT;	(*Object Index 0xA4C0, Subindex 10*)
OutVar11	AT	%QB26.0 : USINT;	(*Object Index 0xA4C0, Subindex 11*)
OutVar12	AT	%QB27.0 : USINT;	(*Object Index 0xA4C0, Subindex 12*)
OutVar13	AT	%QB28.0 : USINT;	(*Object Index 0xA4C0, Subindex 13*)
OutVar14	AT	%QB29.0 : USINT;	(*Object Index 0xA4C0, Subindex 14*)
OutVar15	AT	%QB30.0 : USINT;	(*Object Index 0xA4C0, Subindex 15*)
OutVar16	AT	%QB31.0 : USINT;	(*Object Index 0xA4C0, Subindex 16*)

9.2.3 INT Process Outputs

There are 16 INT values in the 'Process Outputs Area' with the length of 16-bit. The variables holds the very same values like the related CANopen object dictionary entry with index 0xA540.

IEC-61131 declaration example with INT variables:

OutVar1	AT	%QW32.0 : INT;	(*Object Index 0xA540, Subindex 1*)
OutVar2	AT	%QW34.0 : INT;	(*Object Index 0xA540, Subindex 2*)
OutVar3	AT	%QW36.0 : INT;	(*Object Index 0xA540, Subindex 3*)
OutVar4	AT	%QW38.0 : INT;	(*Object Index 0xA540, Subindex 4*)
OutVar5	AT	%QW40.0 : INT;	(*Object Index 0xA540, Subindex 5*)
OutVar6	AT	%QW42.0 : INT;	(*Object Index 0xA540, Subindex 6*)
OutVar7	AT	%QW44.0 : INT;	(*Object Index 0xA540, Subindex 7*)
OutVar8	AT	%QW46.0 : INT;	(*Object Index 0xA540, Subindex 8*)
OutVar9	AT	%QW48.0 : INT;	(*Object Index 0xA540, Subindex 9*)
OutVar10	AT	%QW50.0 : INT;	(*Object Index 0xA540, Subindex 10*)
OutVar11	AT	%QW52.0 : INT;	(*Object Index 0xA540, Subindex 11*)
OutVar12	AT	%QW54.0 : INT;	(*Object Index 0xA540, Subindex 12*)
OutVar13	AT	%QW56.0 : INT;	(*Object Index 0xA540, Subindex 13*)
OutVar14	AT	%QW58.0 : INT;	(*Object Index 0xA540, Subindex 14*)
OutVar15	AT	%QW60.0 : INT;	(*Object Index 0xA540, Subindex 15*)
OutVar16	AT	%QW62.0 : INT;	(*Object Index 0xA540, Subindex 16*)

9.2.4 UINT Process Outputs

There are 16 UINT values in the 'Process Outputs Area' with the length of 16-bit. The variables holds the very same values like the related CANopen object dictionary entry with index 0xA580.

IEC-61131 declaration example with UINT variables:

OutVar1	AT	%QW64.0 : UINT;	(*Object Index 0xA580, Subindex 1*)
OutVar2	AT	%QW66.0 : UINT;	(*Object Index 0xA580, Subindex 2*)
OutVar3	AT	%QW68.0 : UINT;	(*Object Index 0xA580, Subindex 3*)
OutVar4	AT	%QW70.0 : UINT;	(*Object Index 0xA580, Subindex 4*)
OutVar5	AT	%QW72.0 : UINT;	(*Object Index 0xA580, Subindex 5*)
OutVar6	AT	%QW74.0 : UINT;	(*Object Index 0xA580, Subindex 6*)
OutVar7	AT	%QW76.0 : UINT;	(*Object Index 0xA580, Subindex 7*)
OutVar8	AT	%QW78.0 : UINT;	(*Object Index 0xA580, Subindex 8*)
OutVar9	AT	%QW80.0 : UINT;	(*Object Index 0xA580, Subindex 9*)
OutVar10	AT	%QW82.0 : UINT;	(*Object Index 0xA580, Subindex 10*)
OutVar11	AT	%QW84.0 : UINT;	(*Object Index 0xA580, Subindex 11*)
OutVar12	AT	%QW86.0 : UINT;	(*Object Index 0xA580, Subindex 12*)
OutVar13	AT	%QW88.0 : UINT;	(*Object Index 0xA580, Subindex 13*)
OutVar14	AT	%QW90.0 : UINT;	(*Object Index 0xA580, Subindex 14*)
OutVar15	AT	%QW92.0 : UINT;	(*Object Index 0xA580, Subindex 15*)
OutVar16	AT	%QW94.0 : UINT;	(*Object Index 0xA580, Subindex 16*)

9.2.5 DINT Process Outputs

There are 16 DINT values in the 'Process Outputs Area' with the length of 32-bit. The variables holds the very same values like the related CANopen object dictionary entry with index 0xA640.

IEC-61131 declaration example with DINT variables:

OutVar1	AT	%QD96.0 : DINT;	(*Object Index 0xA640, Subindex 1*)
OutVar2	AT	%QD100.0 : DINT;	(*Object Index 0xA640, Subindex 2*)
OutVar3	AT	%QD104.0 : DINT;	(*Object Index 0xA640, Subindex 3*)
OutVar4	AT	%QD108.0 : DINT;	(*Object Index 0xA640, Subindex 4*)
OutVar5	AT	%QD112.0 : DINT;	(*Object Index 0xA640, Subindex 5*)
OutVar6	AT	%QD116.0 : DINT;	(*Object Index 0xA640, Subindex 6*)
OutVar7	AT	%QD120.0 : DINT;	(*Object Index 0xA640, Subindex 7*)
OutVar8	AT	%QD124.0 : DINT;	(*Object Index 0xA640, Subindex 8*)
OutVar9	AT	%QD128.0 : DINT;	(*Object Index 0xA640, Subindex 9*)
OutVar10	AT	%QD132.0 : DINT;	(*Object Index 0xA640, Subindex 10*)
OutVar11	AT	%QD136.0 : DINT;	(*Object Index 0xA640, Subindex 11*)
OutVar12	AT	%QD140.0 : DINT;	(*Object Index 0xA640, Subindex 12*)
OutVar13	AT	%QD144.0 : DINT;	(*Object Index 0xA640, Subindex 13*)
OutVar14	AT	%QD148.0 : DINT;	(*Object Index 0xA640, Subindex 14*)
OutVar15	AT	%QD152.0 : DINT;	(*Object Index 0xA640, Subindex 15*)
OutVar16	AT	%QD156.0 : DINT;	(*Object Index 0xA640, Subindex 16*)

9.2.6 UDINT Process Outputs

There are 16 DINT values in the 'Process Outputs Area' with the length of 32-bit. The variables holds the very same values like the related CANopen object dictionary entry with index 0xA680.

IEC-61131 declaration example with UDINT variables:

OutVar1	AT	%QD160.0 : UDINT;	(*Object Index 0xA680, Subindex 1*)
OutVar2	AT	%QD164.0 : UDINT;	(*Object Index 0xA680, Subindex 2*)
OutVar3	AT	%QD168.0 : UDINT;	(*Object Index 0xA680, Subindex 3*)
OutVar4	AT	%QD172.0 : UDINT;	(*Object Index 0xA680, Subindex 4*)
OutVar5	AT	%QD176.0 : UDINT;	(*Object Index 0xA680, Subindex 5*)
OutVar6	AT	%QD180.0 : UDINT;	(*Object Index 0xA680, Subindex 6*)
OutVar7	AT	%QD184.0 : UDINT;	(*Object Index 0xA680, Subindex 7*)
OutVar8	AT	%QD188.0 : UDINT;	(*Object Index 0xA680, Subindex 8*)
OutVar9	AT	%QD192.0 : UDINT;	(*Object Index 0xA680, Subindex 9*)
OutVar10	AT	%QD196.0 : UDINT;	(*Object Index 0xA680, Subindex 10*)
OutVar11	AT	%QD200.0 : UDINT;	(*Object Index 0xA680, Subindex 11*)
OutVar12	AT	%QD204.0 : UDINT;	(*Object Index 0xA680, Subindex 12*)
OutVar13	AT	%QD208.0 : UDINT;	(*Object Index 0xA680, Subindex 13*)
OutVar14	AT	%QD212.0 : UDINT;	(*Object Index 0xA680, Subindex 14*)
OutVar15	AT	%QD216.0 : UDINT;	(*Object Index 0xA680, Subindex 15*)
OutVar16	AT	%QD220.0 : UDINT;	(*Object Index 0xA680, Subindex 16*)

10 Error Handling

10.1 Programming Environment Error Codes

The programming environment error (and warning) codes will be displayed in a popup window if the programming tool is active. In case of an error the application program will be stopped. Warnings do not stop the application.

Error Code	Description	Comment
1002	Out of program memory Program execution not possible	Program is to big (try with size only)
1004	No valid program	
1005	Download of invalid data	Download incomplete / logical Error
1006	Configuration error / wrong program	
1008	Invalid program number	
1009	Invalid segment number	
1011	Segment already on PLC	
1012	No free watch ID available	Watch table is already full
1013	Invalid command received	
1014	Action not valid. Switch to maintenance first	Operation not allowed in current mode
1015	General network error	Communication error on service interface
1016	Accepted receipt too small	Communication error on service interface
1018	Timer task error	Previous timer task processing was not already finished
1020	Error calling kernel	Error at call of interpreter
1021	Error calling native code	Error at execution of native code
1900	Retain variable handling failed	Too many retain variables or hardware error
1901	NMT boot up error, check CAN configuration	See EPOS P error history for details
1903	One or more slave configuration wrong	Configuration date or time does not match
1904	Problem with persistence memory	Warning only
1905	CAN communication error	See EPOS P error history for details
1908	System was reset by watchdog	Warning only (*)
1909	Interrupt Task error	Previous interrupt processing was not already finished
1911	Execution error: data or program exception	Fatal application processing error
2001	RUN TIME ERROR: division by zero	
2002	RUN TIME ERROR: invalid array index	
2003	RUN TIME ERROR: invalid opcode	Unsupported command
2004	RUN TIME ERROR: opcode not supported	Unsupported command
2005	RUN TIME ERROR: invalid extension	Unsupported command
2006	RUN TIME ERROR: unknown command	Unsupported command
2008	Invalid bit reference	Runtime error
2009	Error restoring data	Runtime error
2010	Invalid array element size	Runtime error
2011	Invalid struct size	Runtime error
2012	RUN TIME ERROR: modulo zero, result undefined	

Table 3: Programming Environment Error Codes

(*): The EPOS Studio uses a watchdog reset also for resetting the Node. Therefore this warning could also be triggered when EPOS Studio manipulates the EPOS P.

10.2 Motion Control Function Blocks Error Codes

The motion control firmware function blocks can return internal error codes as well as error codes (e.g. communication aborts) from the accessed slaves.

Error Code	Description	Comment
0x0000 0000	No error	
0x0000 0001	Internal function block sequence error	
||	Communication abort codes of the connected slave are inserted here (related to DS-301, DSP-402, etc)	Refer to the Firmware Specification of the EPOS
0x0F00 FFC0	The device is in wrong NMT state	
0x0FFF FFF0	CAN communication sequence error	
0x0FFF FFF1	Communication aborted by CAN driver	
0x0FFF FFF2	Communication buffer overflow	
0x0FFF FFF9	Segmented transfer communication error	
0x0FFF FFFA	Wrong axis number	Not in range of 0 .. 31
0x0FFF FFFB	Wrong device number	Not in range of 1..127
0x0FFF FFFC	Wrong CAN port	Not 1 or 2
0x0FFF FFFD	Bad function calling parameters	
0x0FFF FFFE	General CAN communication error	
0x0FFF FFFF	CAN communication time out	

Table 4: Function Block Error Codes

11 Example Projects

11.1 Example «HelloWorld»

Project	HelloWorld
Description	This example project is a very simple project for the first contact with the programming environment. No motion control functionality is used and no motor must be connected. This program can be used to learn the handling of the programming environment and to check the online connection to the EPOS P.
Used Languages	Structured Text
Task	Timer Task (10 ms)
Files	HelloWorld.VAR Project file Counter.ST Main program ReadMe.TXT Additional information

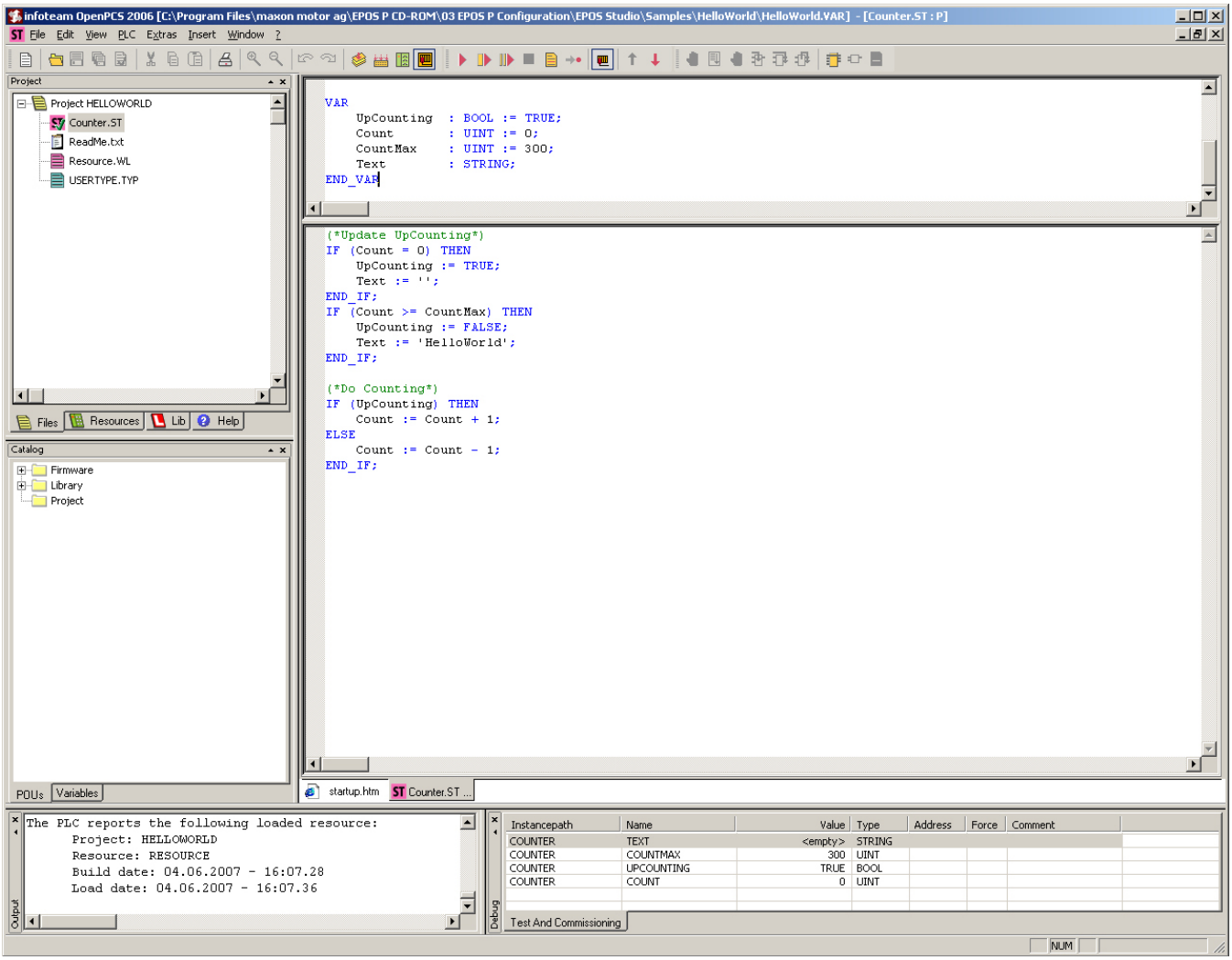


Figure 85: Example ‘HelloWorld’

11.2 Example «SimpleMotionSequence»

Project	SimpleMotionSequence	
Description	The SimpleMotionSequence example consists of two state machines. The first state machine implements the application process and the second implements the error handling. The main state machine moves between two positions. For detailed description of this example, see document «SimpleMotionSequence.pdf».	
Used Languages	SFC (Sequential Function Chart) FBD (Function Block Diagram)	
Task	Cyclic	
Files	SimpleMotionSequence.VAR PROG_Main.SFC PROG_ErrorHandling.SFC	Project file Main program Error handling

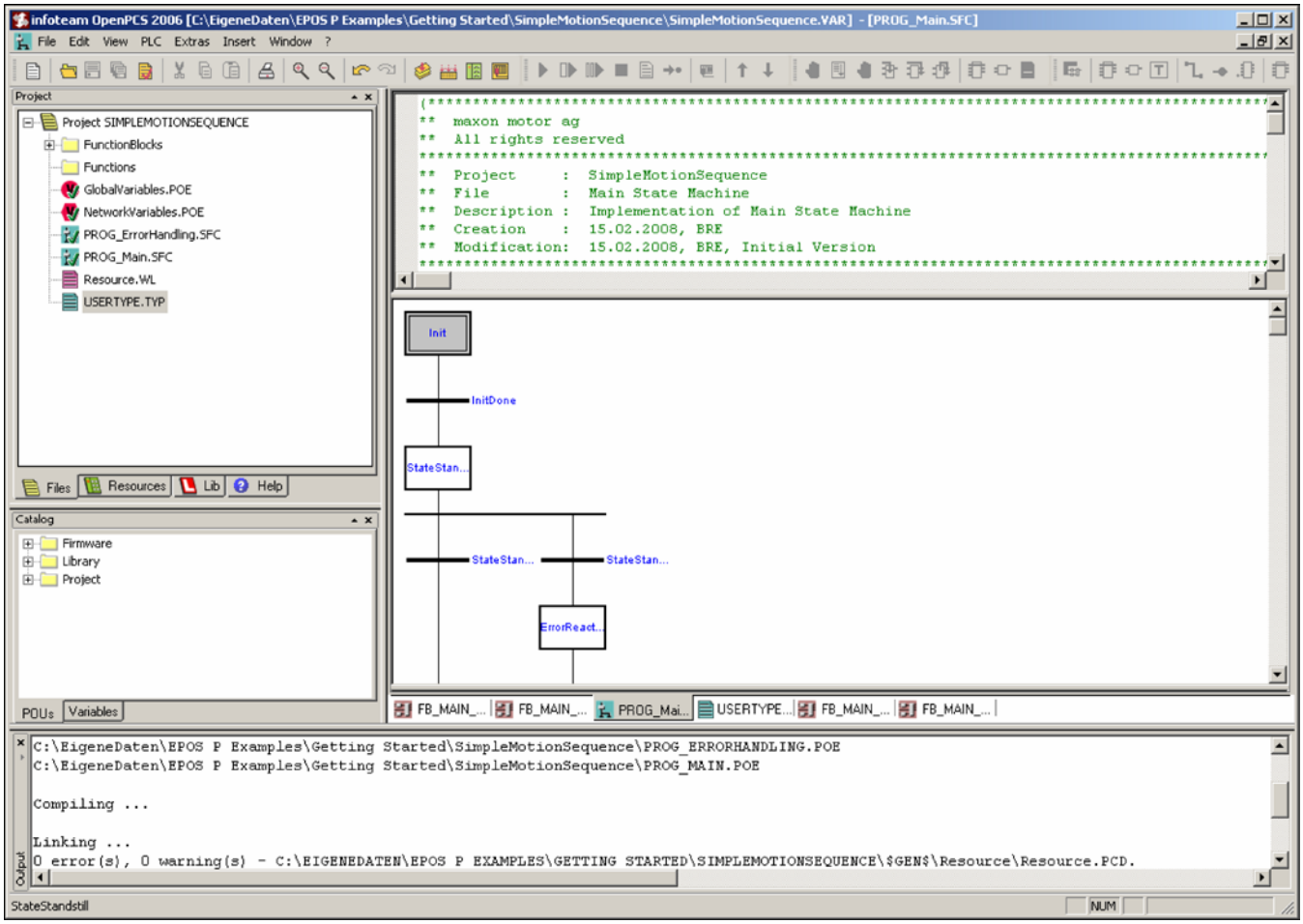


Figure 86: Example ‘SimpleMotionSequence’

11.3 «Best Practice» Program Examples

The «best practice» example collection (available for IEC 611131-3 editors SFC, FBD and ST) shows individual aspects of EPOS P programming. These examples may be a part of a complete application. But they keep focus on single tasks during application programming.

Program example	Description
«State Machine»	The example «State Machine» shows how to implement a state machine including states and transitions. A state machine is the basis and starting point of every EPOS P program. This implementation is the framework for all other examples. For detailed description of this example, see document «StateMachineProject.pdf».
«Error Handling»	The example «Error Handling» demonstrates the usage of the error handling state machine. The state machine detects axis-related errors, communication errors and is gathering error information about the individual error sources. The error information is shown in separate variables on the debug screen. For detailed description of this example, see document «ErrorHandlingProject.pdf».
«Input Output Handling»	The example «Input Output Handling» is used to demonstrate how to read digital and analogue inputs and how to write digital outputs. For detailed description of this example, see document «InputOutputHandlingProject.pdf».
«Homing»	The example «Homing» shows how to configure, start and stop a homing procedure. For detailed description of this example, see document «HomingProject.pdf».
«Positioning»	The example «Positioning» shows how to execute positioning operations. There are three different kinds of positioning presented, two sequential relative positioning, an interrupted positioning and stop-ping the relative positioning. For detailed description of this example, see document «PositioningProject.pdf».
«Continuous Motion»	The example «Continuous Motion» shows how to execute continuous motions. There are three different kinds of continuous motion presented, two sequential continuous motions, an interrupted continuous motion and stopping the continuous motion. For detailed description of this example, see document «ContinuousMotionProject.pdf».
«Actual Value Reading»	The example «Actual Value Reading» shows how to read the actual position, the actual velocity and the actual current of the EPOS. For detailed description of this example, see document «ActualValueReadingProject.pdf».
«Object Dictionary Access»	The example «Object Dictionary Access» shows how to read or write an object from the object dictionary. For detailed description of this example, see document «ObjectDictAccessProject.pdf».
«Data Handling»	The example «Data Handling» shows how to process data. The example is used to read and write bits and to convert data types For detailed description of this example, see document «DataHandlingProject.pdf».

11.4 Application Program Examples

The application example collection shows complete applications of EPOS P programming. These examples may be consisting of some «best practice» examples.

Program example	Description
«Cyclic Motion»	The example «Cyclic Motion» shows typical motion sequences with one axis. Homing, continuous motion and positioning are part of this example. For detailed description of this program example, see document «CyclicMotionProject.pdf».
«I/O Mode»	The example «I/O Mode» shows I/O triggered motions with one axis. For detailed description of this program example, see document «IO_ModeProject.pdf».
«Multi-Axis Motion»	The example «Multi-axis Motion» shows how to implement coordinated motions with two axes. For detailed description of this program example, see document «MultiaxisMotionProject.pdf».

12 Additional Information

[1]	CiA DS-301 Communication Profile for Industrial Systems	www.can-cia.org
[2]	CiA DSP-302 Framework for CANopen Managers and Programmable CANopen Devices	www.can-cia.org (for CiA members only)
[3]	CiA DS-405 Interface and Device Profile for IEC-61131-3 Programmable Devices	www.can-cia.org
[4]	PLCopen: Function blocks for motion control	http://plcopen.org/
[5]	Konrad Etschberger: Controller Area Network (ISBN 3-446-21776-2)	
[6]	maxon motor: EPOS Firmware Specification (Document #798675)	EPOS P CD-ROM or www.maxonmotor.com category <Service & Downloads>
[7]	maxon motor: EPOS P Firmware Specification (Document #810011)	EPOS P CD-ROM or www.maxonmotor.com category <Service & Downloads>