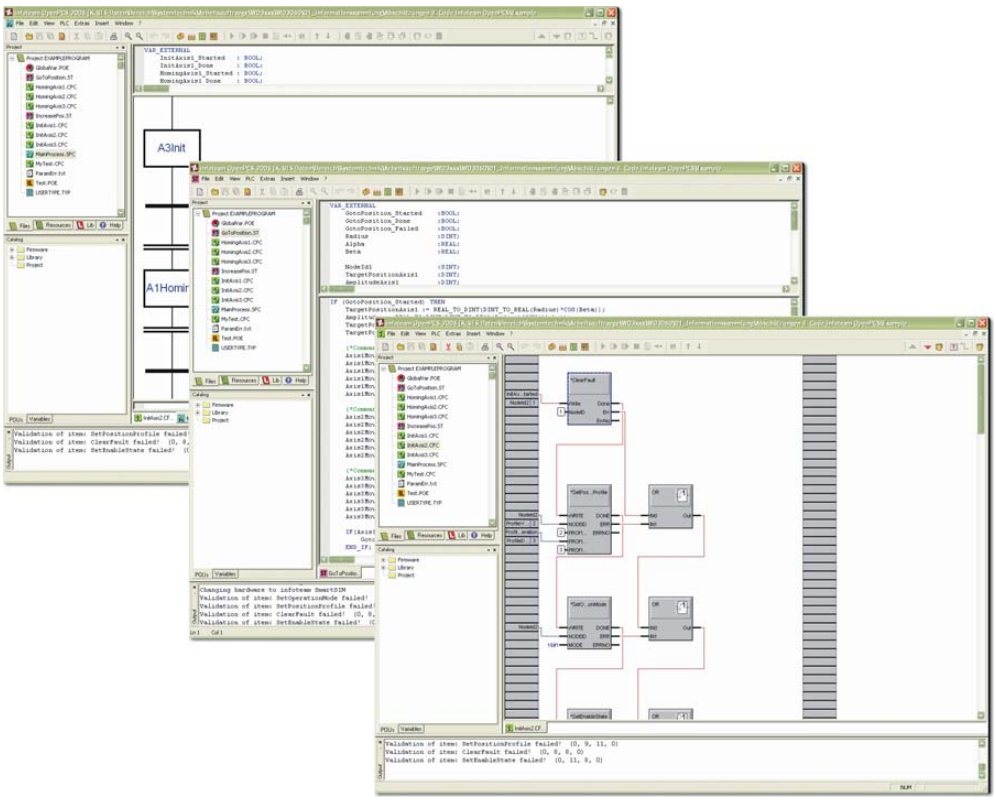
maxon motor control **EPOS P Programmable Positioning Controller**

Programming Reference **Edition January 2010**

EPOS P / MCD EPOS P

Programmable Positioning Controller

Programming Reference



Document ID: 786 912-07

PLEASE READ THIS FIRST

These instructions are intended for qualified technical personnel. Prior commencing with any activities …

- *you must carefully read and understand this manual and*
- *you must follow the instructions given therein.*

We have tried to provide you with all information necessary to install and commission the equipment in a **secure**, **safe** and **time-saving** manner. Our main focus is …

- to familiarize you with all relevant technical aspects,
- to let you know the easiest way of doing,
- to alert you of any possibly dangerous situation you might encounter or that you might cause if you do not follow the description,
- to **write as little** and to **say as much** as possible and
- not to bore you with things you already know.

Likewise, we tried to skip repetitive information! Thus, you will find things **mentioned just once**. If, for example, an earlier mentioned action fits other occasions you then will be directed to that text passage with a respective reference.

Follow any stated reference – observe respective information – then go back and continue with the task!

PREREQUISITES FOR PERMISSION TO COMMENCE INSTALLATION

EPOS P and MCD EPOS P are considered as partly completed machinery according to EU's directive 2006/42/EC, Article 2, Clause (g) and therefore **is intended to be incorporated into or assembled with other machinery or other partly completed machinery or equipment**.

You must not put the device into service, …

- *unless you have made completely sure that the other machinery – the surrounding system the device is intended to be incorporated to – fully complies with the requirements stated in the EU directive 2006/42/EC!*
- *unless the surrounding system fulfills all relevant health and safety aspects!*
- *unless all respective interfaces have been established and fulfill the stated requirements!*

TABLE OF CONTENTS

1 About this Document

1.1 Intended Purpose

The purpose of the present document is to familiarize you with the described equipment and the tasks on safe and adequate installation and/or commissioning.

Observing the described instructions in this document will help you …

- to avoid dangerous situations,
- to keep installation and/or commissioning time at a minimum and
- to increase reliability and service life of the described equipment.

Use for other and/or additional purposes is not permitted. maxon motor, the manufacturer of the equipment described, does not assume any liability for loss or damage that may arise from any other and/or additional use than the intended purpose.

1.2 Target Audience

This document is meant for trained and skilled personnel working with the equipment described. It conveys information on how to understand and fulfill the respective work and duties.

This document is a reference book. It does require particular knowledge and expertise specific to the equipment described.

1.3 How to use

Take note of the following notations and codes which will be used throughout the document.

Notation	Explanation
«Abcd»	indicating a title or a name (such as of document, product, mode, etc.)
¤Abcd¤	indicating an action to be performed using a software control element (such as folder, menu, drop-down menu, button, check box, etc.) or a hardware element (such as switch, DIP switch, etc.)
(n)	referring to an item (such as order number, list item, etc.)
➔	denotes “see”, “see also”, “take note of” or “go to”

Table 1-1 Notations used in this Document

1.4 Symbols and Signs

1.4.1 Safety Alerts

Take note of when and why the alerts will be used and what the consequences are if you should fail to observe them!

Safety alerts are composed of…

- a signal word,
- a description of type and/or source of the danger,
- the consequence if the alert is being ignored, and
- explanations on how to avoid the hazard.

Following types will be used:

1) **DANGER**
   Indicates an **imminently hazardous situation**. If not avoided, the situation will result in death or serious injury.
2) **WARNING**
   Indicates a **potentially hazardous situation**. If not avoided, the situation **can** result in death or serious injury.
3) **CAUTION**
   Indicates a **probable hazardous situation** and is also used to alert against unsafe practices. If not avoided, the situation **may** result in minor or moderate injury.

Example:

DANGER

High Voltage and/or Electrical Shock
Touching live wires causes death or serious injuries!
- *Make sure that neither end of cable is connected to life power!*
- *Make sure that power source cannot be engaged while work is in process!*
- *Obey lock-out/tag-out procedures!*
- *Make sure to securely lock any power engaging equipment against unintentional engagement and tag with your name!*

1.4.2 Prohibited Actions and Mandatory Actions

The signs define prohibitive actions. So, you **must not**!

Examples:

Do not touch!

Do not operate!

The signs point out actions to avoid a hazard. So, you **must**!

Examples:

Unplug!

Tag before work!

1.4.3 Informatory Signs

Requirement / Note / Remark
Indicates an action you must perform prior continuing or refers to information on a particular item.

Best Practice
Gives advice on the easiest and best way to proceed.

Material Damage
Points out information particular to potential damage of equipment.

Reference
Refers to particular information provided by other parties.

1.5 Trademarks and Brand Names

For easier legibility, registered brand names are listed below and will not be further tagged with their respective trademark. It must be understood that the brands (the below list is not necessarily concluding) are protected by copyright and/or other intellectual property rights even if their legal trademarks are omitted in the later course of this document.

The brand name(s) …	… is/are a registered trademark(s) of …
Adobe® Reader®	© Adobe Systems Incorporated, USA-San Jose, CA
Pentium®	© Intel Corporation, USA-Santa Clara, CA
Windows®	© Microsoft Corporation, USA-Redmond, WA

Table 1-2 Brand Names and Trademark Owners

1.6 Sources for additional Information

For further details and additional information, please refer to below listed sources:

#	Reference
[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 (section accessible 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»

Table 1-3 Sources for additional Information

1.7 Copyright

maxon motor ag
Brünigstrasse 220
P.O.Box 263
CH-6072 Sachseln
Switzerland

Phone +41 (41) 666 15 00
Fax +41 (41) 666 15 50

www.maxonmotor.com

2 Introduction

2.1 Important Notice: Prerequisites for Permission to commence Installation

EPOS P and MCD EPOS P are considered as partly completed machinery according to EU's directive 2006/42/EC, Article 2, Clause (g) and therefore **is only intended to be incorporated into or assembled with other machinery or other partly completed machinery or equipment**.

WARNING

Risk of Injury

Operating the device without the full compliance of the surrounding system with the EU directive 2006/42/EC may cause serious injuries!

- *Do not operate the device, unless you have made sure that the other machinery fulfills the requirements stated in EU's directive!*
- *Do not operate the device, unless the surrounding system fulfills all relevant health and safety aspects!*
- *Do not operate the device, unless all respective interfaces have been established and fulfill the stated requirements!*

2.2 General Information

The present document provides you with information on programming the EPOS P and MCD EPOS P Programmable Positioning Controller. It describes the standard procedure to write and debug an IEC 61131 program based on an example and describes motion control function blocks.

Find the latest edition of the present document, as well as additional documentation and software to the EPOS P and MCD EPOS P Programmable Positioning Controller also on the internet:

➔www.maxonmotor.com – category «Service & Downloads»

➔shop.maxonmotor.com

2.3 Documentation Structure

The present document is part of a documentation set. Please find below an overview on the documentation hierarchy and the interrelationship of its individual parts:

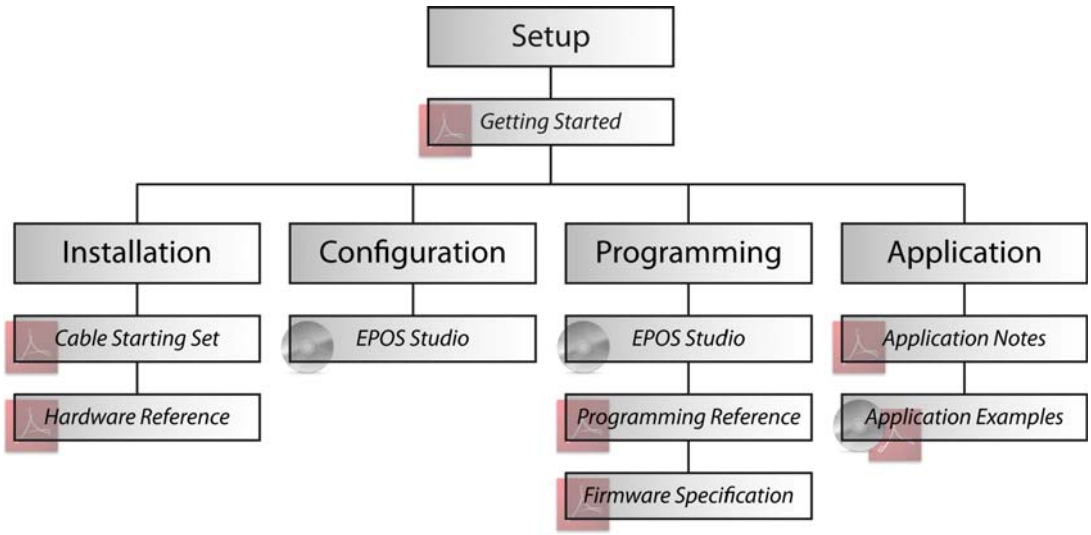


Figure 2-1 Documentation Structure

2.4 Safety Precautions

Prior continuing …

- make sure you have read and understood the section “PLEASE READ THIS FIRST” on page A-2,
- do not engage with any work unless you possess the stated skills (➔chapter “1.2 Target Audience” on page 1-5,
- refer to section “Symbols and Signs” on page 1-6 to understand the subsequently used indicators,
- you must observe any regulation applicable in the country and/or at the site of implementation with regard to health and safety/accident prevention and/or environmental protection,
- take note of the subsequently used indicators and follow them at all times.

DANGER

High Voltage and/or Electrical Shock
Touching live wires causes death or serious injuries!

- *Consider any power cable as connected to life power, unless having proven the opposite!*
- *Make sure that neither end of cable is connected to life power!*
- *Make sure that power source cannot be engaged while work is in process!*
- *Obey lock-out/tag-out procedures!*
- *Make sure to securely lock any power engaging equipment against unintentional engagement and tag with your name!*

Requirements

- *Make sure that all associated devices and components are installed according to local regulations.*
- *Be aware that, by principle, an electronic apparatus can not be considered fail-safe. Therefore, you must make sure that any machine/apparatus has been fitted with independent monitoring and safety equipment. If the machine/apparatus should break down, if it is operated incorrectly, if the control unit breaks down or if the cables break or get disconnected, etc., the complete drive system must return – and be kept – in a safe operating mode.*
- *Be aware that you are not entitled to perform any repair on components supplied by maxon motor.*

Best Practice

- *For initial operation, make sure that the motor is free running. If not the case, mechanically disconnect the motor from the load.*

Maximal permitted Supply Voltage

- *Make sure that supply power is between 11…24 VDC.*
- *Supply voltages above 27 VDC will destroy the unit.*
- *Wrong polarity will destroy the unit.*

Electrostatic Sensitive Device (ESD)

- *Make sure to wear working cloth in compliance with ESD countermeasures.*
- *Handle device with extra care.*

3 Programming

3.1 Programming Tool «OpenPCS»

3.1.1 Startup

1) Open «EPOS Studio».
2) Load a project (*.pjm), containing a programmable controller (can be an EPOS P or MCD?EPOS?P) permitting you to open the programming tool.
3) Click page ¤Tools¤ in page navigator.

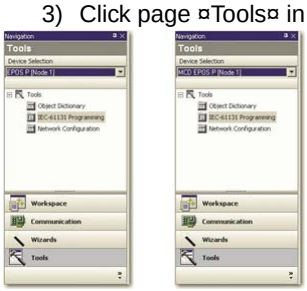


Figure 3-2 Page Navigator

4) Select desired device in device selection combo box.
5) Doubleclick ¤IEC 61131 Programming¤. A list of sample projects will be displayed. Use this view as a “control center” to open projects and control program status (for details ➔Table 3-4).

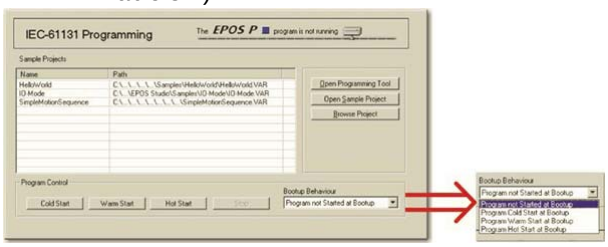


Figure 3-3 EC 61131 Programming Window

Area	Button / Command	Effect
Project	¤Open Programming Tool¤	Launches external tool «OpenPCS»
	¤Open Sample Project¤	Opens an existing project
	¤Browse Project¤	Searches for/opens an existing IEC 61131 project (*.var)
Program Control	¤Cold Start¤	Starts the program from scratch by initializing variables to their default values
	¤Warm Start¤	Restarts the program and restores the values
	¤Hot Start¤	Restarts the program at the position it was stopped and restores values
	¤Stop¤	Interrupts the program
Bootup Behavior	¤Bootup Behavior¤	Defines properties after download or reset of program

Table 3-4 EC 61131 Programming Window – Commands and their Effect

6) Click ¤Open Programming Tool¤ to open external tool «OpenPCS».

3.1.2 Licence Key Configuration

In order to use the programming tool «OpenPCS», a valid licence key must be configured.

1) Open menu ¤Extras¤, then submenu ¤Tools¤.
2) Click ¤Info¤ to check if valid license is available.
   If no license is registered, enter valid serial number and license key (➔"ReadMe.txt" in EPOS Studio directory).

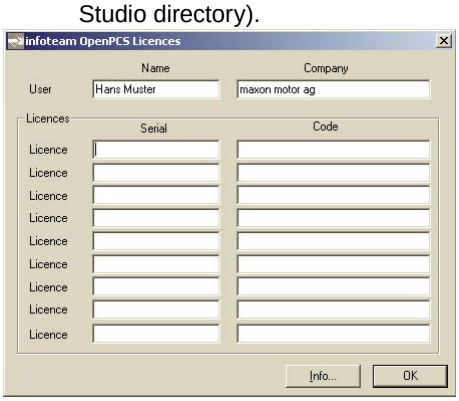


Figure 3-4 OpenPCS License Registration

If you find the license key out of date, download latest version of «EPOS Studio» from the internet (for URLs ➔chapter "2 Introduction" on page 2-9.

3.2 Connection Setup

1) Open menu ¤PLC¤, then click menu item ¤Connections¤.

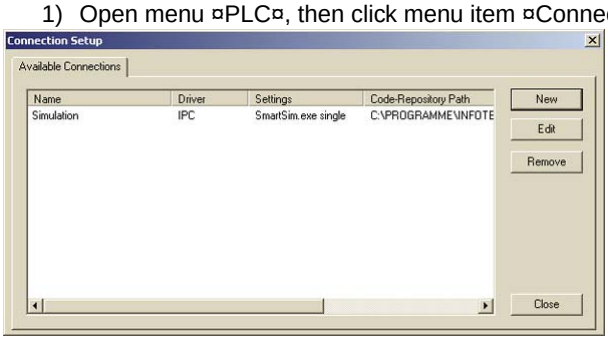


Figure 3-5 Connection Setup

2) Look for entry “ProxyEpos”:
   a) **If available**, click ¤Edit¤ and continue with step 5.
   b) **If not available**, click ¤New¤ and continue with next step.
3) Enter “ProxyEpos” as name and add comments – later on, this driver will enable parallel communication of «EPOS Studio» and programming tool «OpenPCS». Then click ¤OK¤.

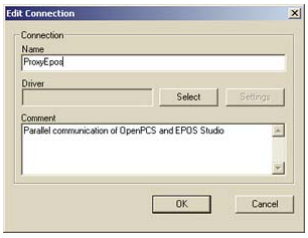


Figure 3-6 Edit Connection

4) Click ¤OK¤ to select driver “ProxyEpos”.

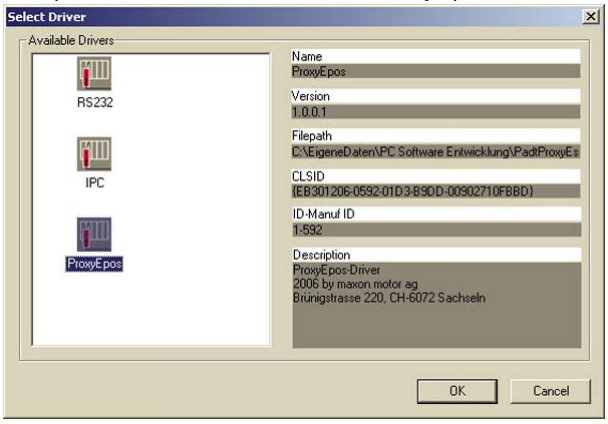


Figure 3-7 Select Driver

5) Click ¤Settings¤ and select baud rate of your RS232 interface. Then click ¤OK¤.

Please note: Default baud rate is 115200 Bd.

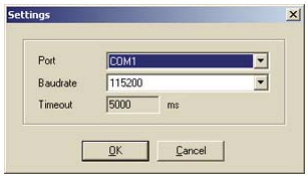


Figure 3-8 Connection Settings

6) The connection entry has been added to the list and is available for selection.

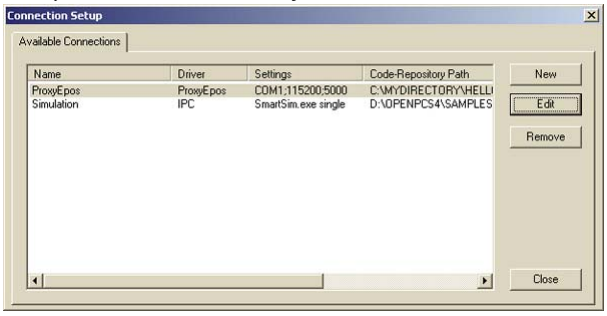


Figure 3-9 Connection Entry “ProxyEpos”

Document ID: 786 912-07 V1 en
Edition: January 2010

3.3 Sample Project «HelloWorld»

The following chapters explain the standard procedure to write a program.

The procedure is described using an example of a very simple program without any motion control features. The intention of this program is only to visualize handling of the programming tool. Basically, the program counts up and down. When reaching the maximum value, the text “HelloWorld” will be written to the variable “Text”.

For an example using motion control functionality ➔chapter “9.1 «HelloWorld»” on page 9-91.

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
```

3.4 Creating New Project

1) Click menu ¤File¤, then submenu ¤Projects¤. Select menu item ¤New¤.
2) Select file type ¤maxon motor ag¤ and template ¤EPOS P Project¤ or ¤MCD EPOS P Project¤.
3) Enter project name “HelloWorld”, browse for location to store new project.
4) Click ¤OK¤ to create new project. It will contain a resource item containing configuration for the hardware module named “maxon motor EPOS P” and a network connection named “ProxyEpos”.

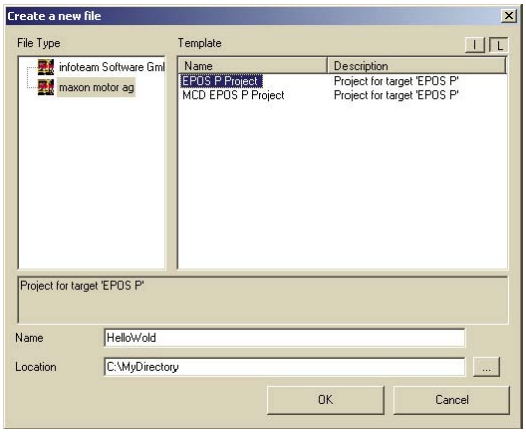


Figure 3-10 Create New Project

5) To view/edit resource specification, click menu ¤PLC¤, then menu item ¤Resource Properties¤.

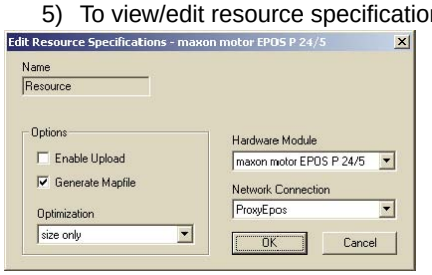


Figure 3-11 Edit Resource Specifications

Document ID: 786 912-07 V1 en
Edition: January 2010

3.5 Program Code

3.5.1 Writing Program Code

1) Add a new program to the project:
   a) Click menu ¤File¤, then menu item ¤New¤ to open dialog.
   b) Select ¤Create a new file¤.

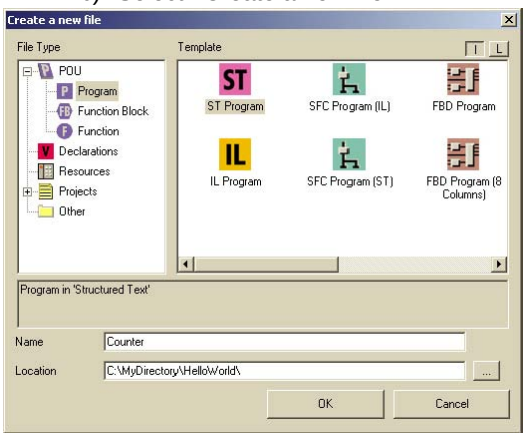


Figure 3-12 Create Program File

2) Select file type ¤Program¤ from directory “POU” (Program Organization Unit):
   a) Choose preferred programming language for your program – in following example “Structured Text”.
   b) Enter name “Counter” and click ¤OK¤.
3) You will be asked whether or not you wish to add program item “Counter” to the active resource. Click ¤Yes¤.

Figure 3-13 Add to active Resource

4) Configure configuration of program “Counter”:
   a) Open tab ¤Resources¤, select task item ¤Counter¤ and open properties via context menu (right click).
   b) Select task type ¤Timer¤ and set time to 10 ms.

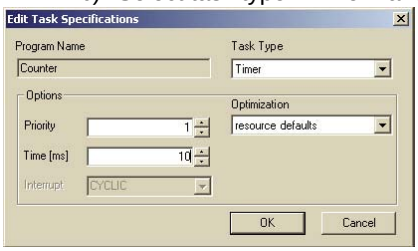


Figure 3-14 Task Specifications

5) Now, you are ready to start programming:

a) Open program item ¤Counter.ST¤.

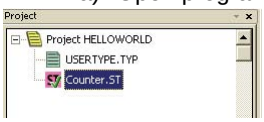


Figure 3-15 Project HelloWorld

b) Enter 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 3-16 Variable Declaration

c) Enter 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 3-17 Program Code

6) Verify correct implementation:
Click menu ¤File¤, then select menu item ¤Check Syntax¤.

3.5.2 Compiling and executing Program Code

1) After code implementation, the program must be compiled:
Click menu ¤PLC¤, then select menu item ¤Build Active Resource¤. The following logging output will be displayed.

```
C:\Programme\infoteam Software\OpenPCS2006\\itmake.exe -m "C:\MyDirectory\HelloWorld\$ENV$\Resource\Resource.MAK
Building resource C:\MyDirectory\HelloWorld\$ENV$\Resource\Resource.MAK.
Executing Pre-Build-Steps:

Creating dependency list(s):
C:\MyDirectory\HelloWorld\Counter.POE

Compiling ...

Linking ...
0 error(s), 0 warning(s) - C:\MYDIRECTORY\HELLOWORLD\$GEN$\Resource\Resource.PCD.

Executing Post-Build-Steps:
Total:
0 error(s) 0 warning(s)
```

Output

Figure 3-18 Output Window

2) In order to download the program code, an online connection must be established:
   a) Click menu ¤PLC¤, then select menu item ¤Online¤.
   b) If new code is detected, you will be asked whether or not you wish to download the current resource. Click ¤Yes¤ to update the program in EPOS P.

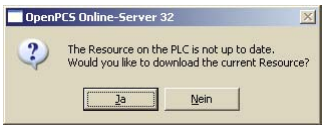


Figure 3-19 Download new Code

3) Click menu ¤PLC¤, then select menu item ¤Cold Start¤ to start downloaded code.

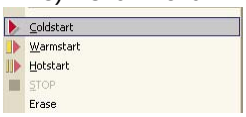


Figure 3-20 Cold Start

3.5.3 Debugging Program Code

1) Add a watch variable to the window “Test And Commissioning”:
   a) Open tab ¤Resources¤ in the project window.
   b) Open tree view of task ¤COUNTER¤ and select variable ¤COUNT¤.
   c) Select command ¤Add To Watchlist¤ from context menu. The variable “COUNT” will now be added to window “Test And Commissioning”.

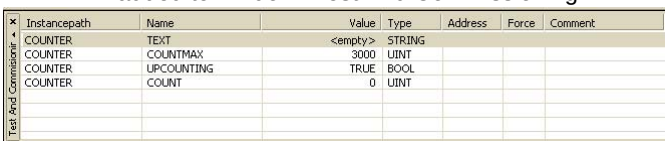


Figure 3-21 “Test And Commissioning” Window

2) Repeat above procedure for variables “UPCOUNTING” and “COUNTMAX”.
3) For a step-by-step program debugging add a breakpoint to the program code:
   a) Position mouse cursor to the line you wish to add the breakpoint.
   b) Click menu ¤PLC¤, then submenu ¤Breakpoint¤ and select menu item ¤Toggle¤. The program will then stop at the breakpoint.

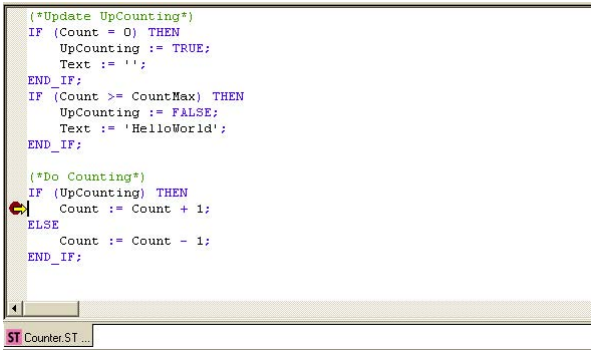


Figure 3-22 Adding a “Breakpoint”

4) To delete a breakpoint, again toggle the breakpoint.

5) Continue program execution:
   a) Click menu ¤PLC¤, then submenu ¤Breakpoint¤.
   b) Select menu item ¤Go¤.

Figure 3-23 Continue Program Execution

4 Project Settings

The following chapter will explain functions of some project-specific settings that need to be performed during the programming process.

4.1 Resource Properties

In general, a resource is equivalent to a PLC or a micro controller. A resource definition consists of…

- name (for identification),
- hardware description (i.e. information on properties of your PLC used by «OpenPCS»), and
- a connection name (i.e. information on type of communication between «OpenPCS» and the control system).

A resource maintains a list of tasks which will be run on the control system.

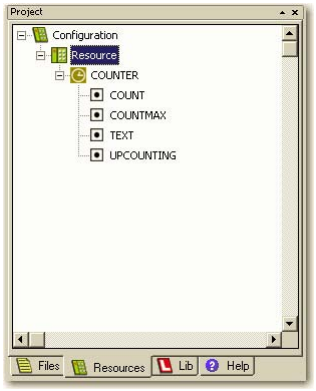


Figure 4-24 Resource Pane

4.1.1 Edit Resource Properties

Right click to open context menu and select ¤Properties¤. A dialog box will be displayed permitting you to change the following properties:

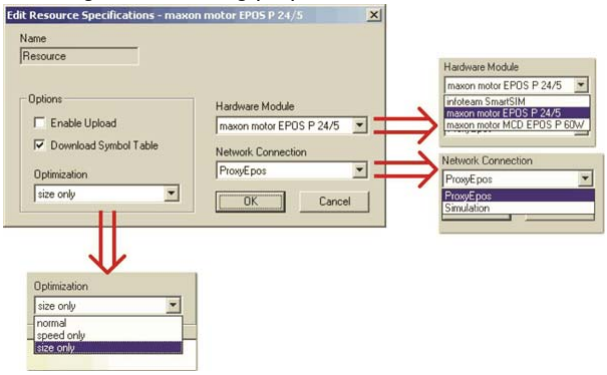


Figure 4-25 Resource Specifications Window

Control Element	Description
Hardware Module	Select the configuration file corresponding to the controller you are using. When using maxon hardware, the modules “maxon motor EPOS P 24/5” and “maxon motor MCD EPOS P 60 W” will be available. If you wish to use Windows SmartSIM simulation, use “SmartSIM”.
Network Connection	Select the communication connection 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 to optimize the generated code in respect to its size **speed only**: compiler option to optimize the generated code in respect to speed **normal**: mix between size only and speed only

Table 4-5 Resource Specifications Window – Control Elements

Remark
Bear in mind that full debugging is only possible with optimization option “size” only!

4.2 Task Properties

In general, a task is equivalent to a program. The definition of a task consists of…

- name,
- information on the execution of the task, and
- POU of type PROGRAM that will be executed in this task.

4.2.1 Edit Task Properties

Right click to open context menu and select ¤Properties¤. A dialog box will be displayed permitting you to change the following properties.

4.2.1.1 Task Type

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

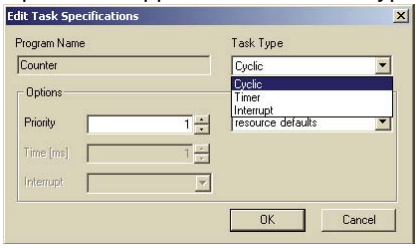


Figure 4-26 Task Type Window

Control Element	Description
Cyclic	Will be executed when no timer or interrupt tasks are ready to run. The priority (may be specified in 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	Will be executed every n milliseconds (n may be specified in task properties).
Interrupt	Will be executed as soon as the interrupt occurs to which they are linked to.

Table 4-6 Task Type Window – Control Elements

4.2.1.2 Optimization

OpenPCS supports optimization settings “speed”, “size” and “normal”.

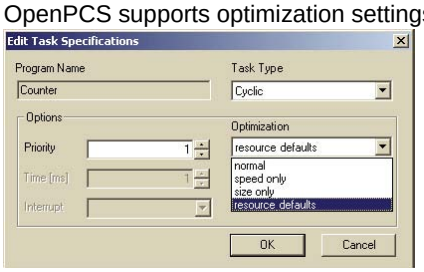


Figure 4-27 Edit Task Specification – Optimization

Control Element	Description
resource defaults	Uses the optimization attributes of the resource.
size only	Compiler option to optimize the generated code in respect to its size.
speed only	Compiler option to optimize the generated code in respect to speed.
normal	Mix between size only and speed only.

Table 4-7 Edit Task Specification – Control Elements

Remark

Bear in mind that full debugging is only possible with optimization option “size” only!

4.2.1.3 Interrupt

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

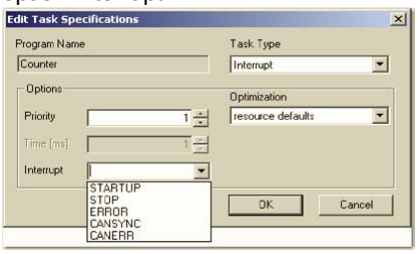


Figure 4-28 Edit Task Specification – Interrupt

Control Element	Description
STARTUP	Task with type interrupt is executed once upon startup.
STOP	Task with type interrupt is executed once upon program stop.
ERROR	Task with type interrupt is executed once upon program error.
CANSYNC	Task with type interrupt is synchronized with CANopen SYNC.
CANERR	Task with type interrupt is synchronized with CANopen EMCY.

Table 4-8 Edit Task Specification – Control Elements

Remark

- *Function Blocks "STARTUP", "STOP" and "ERROR" need typically more than one cycle to finish!*
- *Function Block "CANSYNC": The interrupt source for this task is the CANopen SYNC Cycle, the task will never be called when the SYNC Master is not activated*
- *Function Block "CANERR": 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.*

4.3 Network Configuration

Used to setup a multi axes network. Use this tool to configure the devices (master and slaves) that will be used in a multi axis IEC 61131 program.

4.3.1 Overview

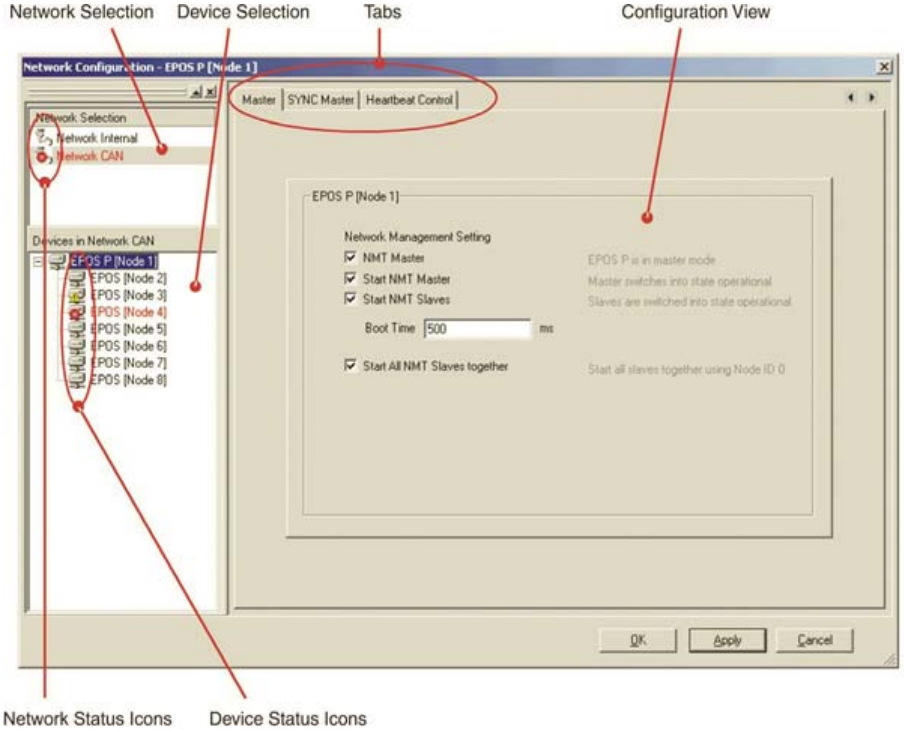


Figure 4-29 Network Configuration Overview

Control Element	Description
Network Selection	Displays all available networks.
Device Selection	Displays all available devices in the network selected.
Tabs	Selection of configuration views.
Configuration View	Configuration view to change settings.

Table 4-9 Network Configuration Overview – Control Elements

Status	Icon	Description
Network Status	OK	No error or warning in this network.
	Warning	No warnings in this network. Check devices.
	Error	No errors in this network. Check devices.
Device Status	OK	No error or warning in this device configuration.
	Warning	No warnings in this device configuration. Check configuration views.
	Error	No errors in this device configuration. Check configuration views.

Table 4-10 Network Configuration Overview – Status & Icons

4.3.2 Master Configuration

For the master configuration, select the master item in the device selection. The master must be configured or all networks.

4.3.2.1 Configuration View “Master”

The configuration view “Master” allows definition of behavior of the master device.

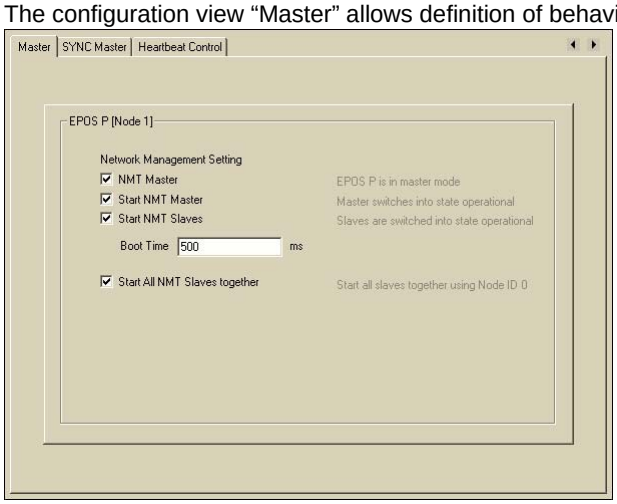


Figure 4-30 Configuration View “Master”

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.
NMT Slaves together	Checked	All slaves are starting at the same time using a broadcast service.

Table 4-11 Configuration View “Master” – Options and Defaults

4.3.2.2 Configuration View “SYNC Master”

Allows definition of behavior of the SYNC Master in the network. The SYNC Master must be active if any synchronous PDO is being configured.

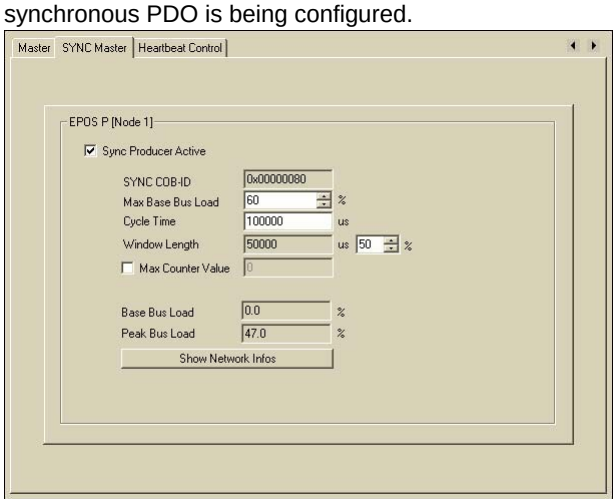


Figure 4-31 Configuration View “SYNC Master”

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 data byte containing a counter value.

Table 4-12 Configuration View "SYNC Master" – Options and Defaults

Calculations	Description
Base Bus Load	Calculated bus load containing CAN frames that are cyclically transmitted. Following CAN frames are included in calculation: SYNC, PDO sync, Heartbeat.
Peak Bus Load	Calculated bus load containing all CAN frames that are transmitted. Following CAN frames are included: SYNC, PDO sync, Heartbeat, PDO async. **Note:** Asynchronous PDOs are a potential risk for bus overload. Use "Inhibit Time" to limit the transmission rate.

Table 4-13 Configuration View "SYNC Master" – Calculations

Best Practice: How to reduce Bus Load

If bus load exceeds the maximum bus load, the transmission of CAN frames must be limited. Use one of the following actions to reduce the bus load.

Action	Object	Description / Effect
Increase CAN Bitrate	all	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 PDOs. Increasing the inhibit time is reducing the update rate of network variables in your IEC 61131 program.

Table 4-14 Configuration View "SYNC Master" – Best Practice

For more details click ¤Show Network Infos¤:

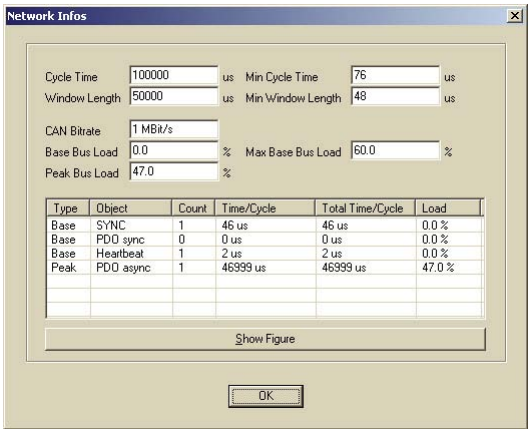


Figure 4-32 Network Info

Parameter	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	Minimum Window Length calculated based on the maximum base bus load.
CAN Bitrate	Configured CAN Bitrate.
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 PDOs are a potential risk for a bus overload. Use the “Inhibit Time” to limit the transmission rate

Table 4-15 Network Info – Parameters

Parameter	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 PDOs 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.

Table 4-16 Network Info – Table Columns

Click ¤Show Figure¤ to display timing diagram:

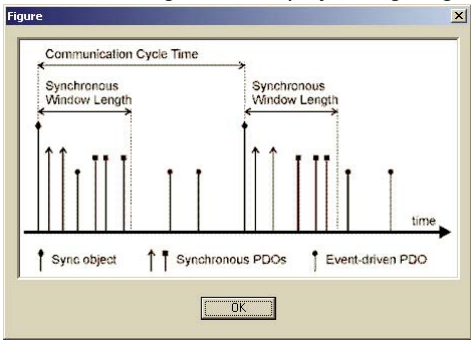


Figure 4-33 Cycle Time

4.3.2.3 Configuration View “Heartbeat Control”

Allows definition of the error control behavior 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.

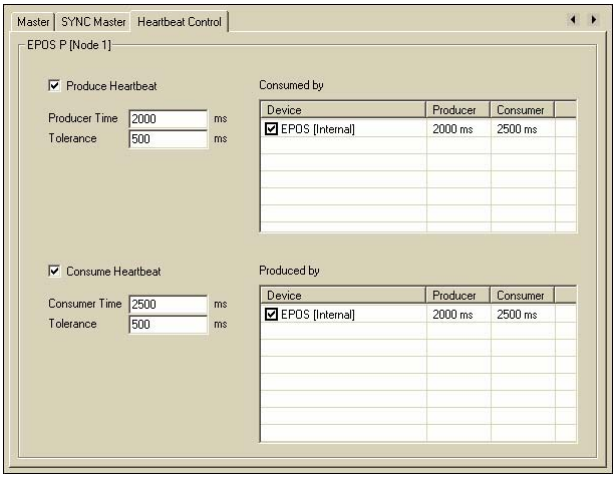


Figure 4-34 Configuration View “Heartbeat Control”

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

Table 4-17 Configuration View “Heartbeat Control” – Options and Defaults Producer

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

Table 4-18 Configuration View “Heartbeat Control” – Options and Defaults Consumer

4.3.3 Slave Configuration

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

4.3.3.1 Configuration View “Slave”

Allows to define the behavior of the slave device.

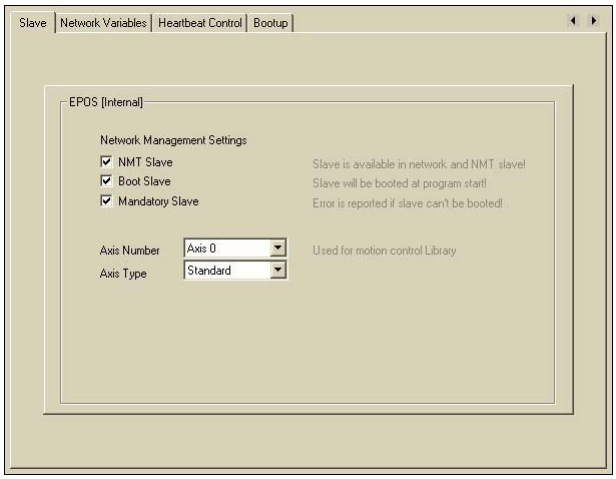


Figure 4-35 Configuration View “Slave”

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. **Note:** 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. **Note:** If the axis type is not defined as “Standard”, the motion control function blocks can’t be used.

Table 4-19 Configuration View “Slave” – Options and Defaults

4.3.3.2 Tab “Network Variables”

Allows to setup network variables for the IEC 61131 program.

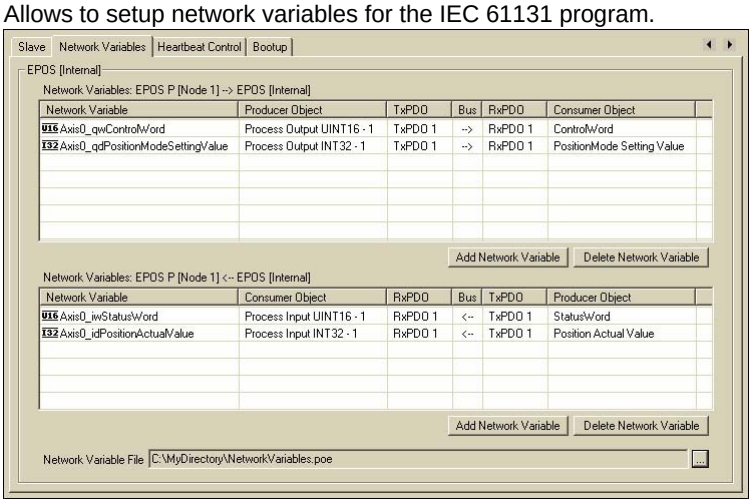


Figure 4-36 Tab “Network Variables”

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 4-20 Network Variables: EPOS P [Node 1] to EPOS [Internal]

Network Variables: EPOS P [Node 1] ← EPOS [Internal]

Displays all configured network variables sent from the slave to the master.

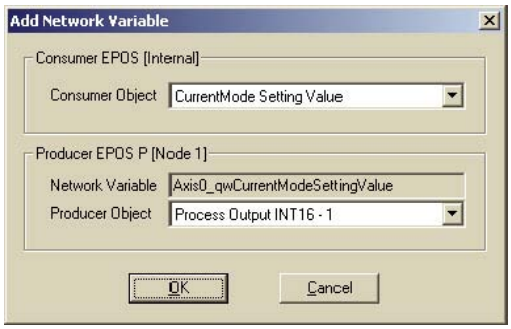
Column	Description
Network Variable	Name of network variable to be used in IEC 61131 program. The network variables can be exported to a network variable file (*.poe).
Consumer Object	Object in object dictionary of the master. This object is mapped to the receive PDO.
RxPDO	Configured receive PDO to 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.

Table 4-21 Network Variables: EPOS P [Node 1] from EPOS [Internal]

Add Network Variable

Either click ¤Add Network Variable¤ or right click 

.



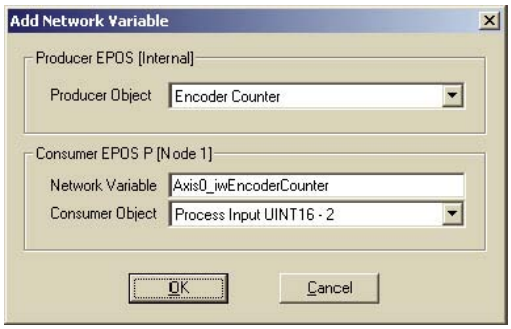


Figure 4-37 Add Network Variable

Direction	Parameter	Description
Master → Slave	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.
Slave → Master	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.

Table 4-22 Add Network Variable – Parameters

Delete Network Variable

Either click ¤Delete Network Variable¤ or right click 

.

Edit PDO Links

PDO links are automatically created when adding a new network variable. Edit them using right click 

.

The dialog “Edit PDO Links” shows all PDOs linked between the master and the slave device. The configuration of the PDO can be changed using this dialog.

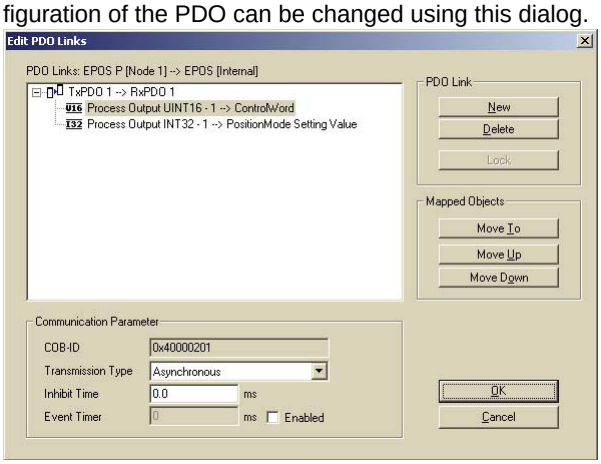


Figure 4-38 Edit PDO Links

Communication Parameter

Parameter	Description
COB-ID	COB-ID of the linked PDOs.
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 PDOs. **Note:** 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.

Table 4-23 Edit PDO Links – Communication Parameter

PDO Link

Control Element	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.

Table 4-24 Edit PDO Links – PDO Link

Mapped Objects

Control Element	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.

Table 4-25 Edit PDO Links – Mapped Objects

Lock or Unlock PDOs

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

Right click 

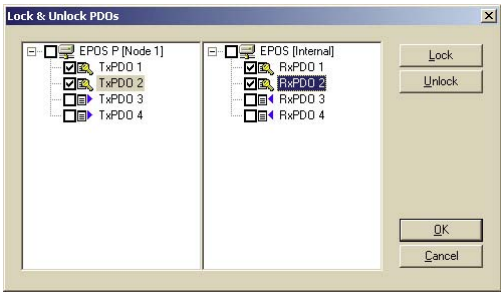


Figure 4-39 Lock/unlock PDOs

Icon	Description
Locked PDO	Can not 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.

Table 4-26 Lock or Unlock PDOs – Icons

Reset PDOs

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

Right click 

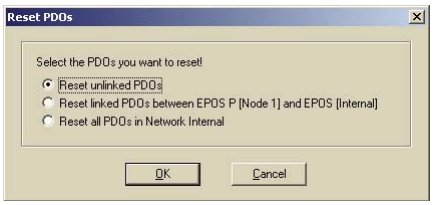


Figure 4-40 Reset PDOs

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

Table 4-27 Reset PDOs – Options

Show Network Variable File

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

Right 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.

Right click 

Print Network Variable File

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

Right 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 4-41 Declaration of Network Variables

4.3.3.3 Configuration View “Heartbeat Control”

Allows definition of error control behavior 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.

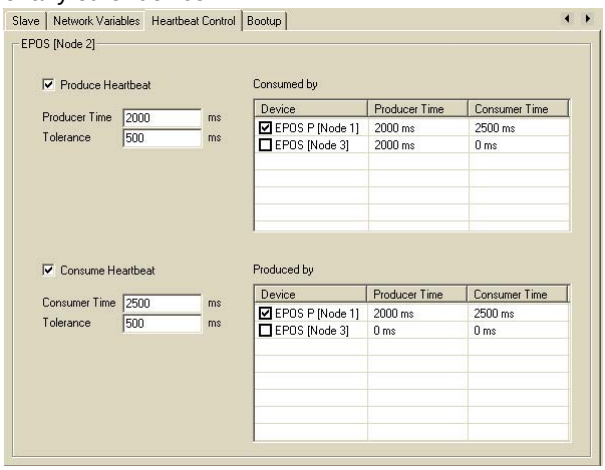


Figure 4-42 Configuration View “Heartbeat Control”

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 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

Table 4-28 Configuration View “Heartbeat Control” – Options and Defaults Producer

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

Table 4-29 Configuration View “Heartbeat Control” – Options and Defaults Consumer

4.3.3.4 Configuration View “Bootup”

Allows definition of 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.

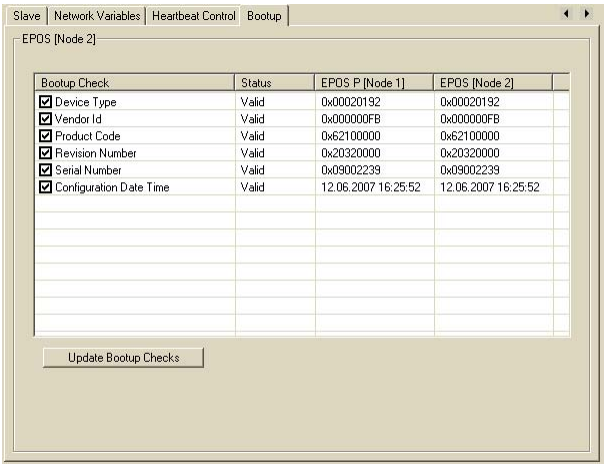


Figure 4-43 Configuration View “Bootup”

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.

Table 4-30 Configuration View “Bootup” – Options and Defaults Consumer

4.3.4 Minimal Network Configuration

In order to use a motion control axis in a IEC 61131 program, the following configuration steps will be necessary.

1) **Step 1: Create Project in EPOS Studio**
   a) Select menu item ¤New Project¤ in menu “File”.
   b) Select an EPOS P project template and click ¤Next¤.
   c) Enter project name, destination directory and click ¤Finish¤.
2) **Step 2: Scan the Network Topology**
   a) Change to tab ¤Communication¤ in navigation window.
   b) Select icon for CAN network and execute command “Scanning Devices” in context menu.
   c) Enter scanning settings.
   d) Start Scanning.
   e) Click ¤OK¤ to close dialog “Scanning Devices”.
   f) Connect all new scanned devices.
3) **Step 3: Open the Tool “Network Configuration”**
   a) Change to tab ¤Tools¤ in navigation window.
   b) Select device ¤EPOS P¤ in device selection.
   c) Click item ¤Network Configuration¤ to open tool.
4) **Step 4: Minimal Master Configuration**
   a) Select master device ¤EPOS P¤ in device selection.
   b) Select configuration view “Master” and configure following options:
      • NMT Master: Enabled
      • Start NMT Master: Enabled
      • Start NMT Slaves: Enabled
      • Boot Time: 500 ms
      • Start All NMT Slaves together: Enabled
   c) Select configuration view “SYNC Master” and disable Sync Producer.
   d) Select configuration view “Heartbeat Control” and disable Heartbeat Producer.
5) **Step 5: Minimal Slave Configuration**
   a) Select one of the slave devices in device selection.
   b) Select configuration view “Slave” and configure 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
   c) Select configuration view “Heartbeat Control” and disable Heartbeat Producer.
   d) Select configuration view “Booting” and disable all bootup checks.
   e) Repeat slave configuration for all slaves in your system.
6) **Step 6: Save Network Configuration**
   Click ¤OK¤ to save network configuration.
7) **Step 7: Start writing your IEC 61131 program**
   Open programming tool and write your program addressing network devices.

4.3.4.1 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”:	

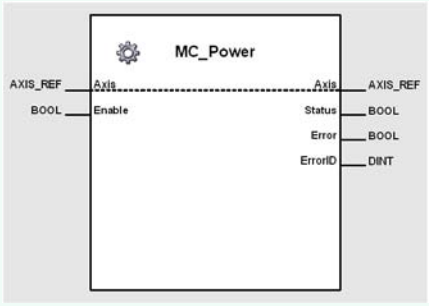
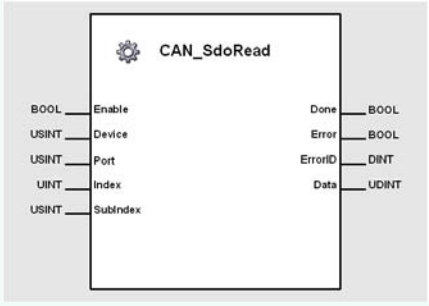
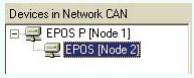


Table 4-31 Motion Control Function Block: Configuration of Axis Number

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

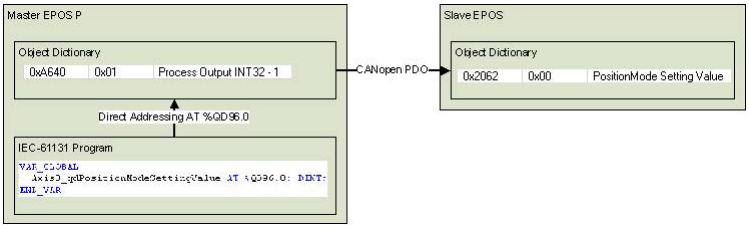
Table 4-32 CANopen DS-301 Function Block: Configuration of Node ID

4.3.4.2 Communication via Network Variables

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

1) **Step 1: Open Configuration View “Network Variables”**
   a) Open tool “Network Configuration”.
   b) Select one of the slave devices in device selection and activate configuration view “Network Variables”
2) **Step 2: Define Output Network Variables**
   Network Variables from the master to the slave can be used to control a slave device.
   a) Click ¤Add Network Variable¤ in the upper part of the view.
   b) Select a consumer object in selection combo box.
   c) Click ¤OK¤ to confirm selection.
   d) Repeat steps for each network variable

Network Variable from IEC 61131 Program to Slave



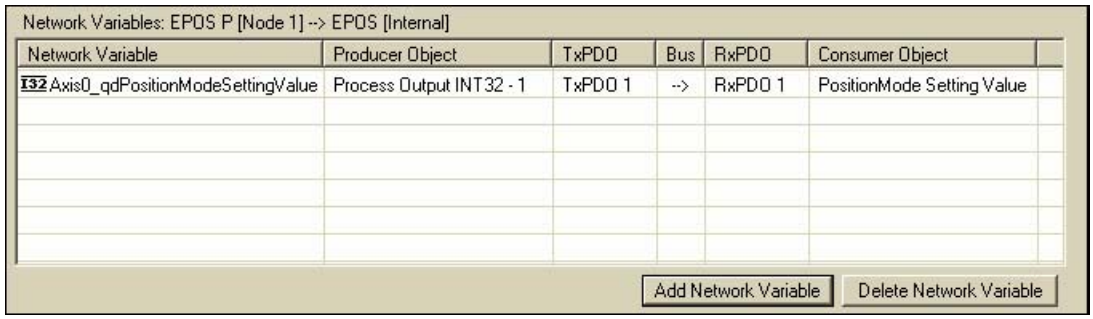
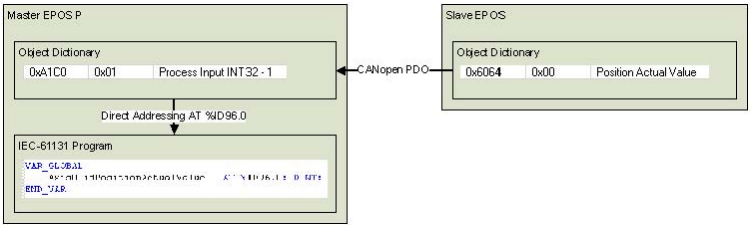


Figure 4-44 Output Network Variables

3) **Step 3: Define Input Network Variables**
Network Variables from the slave to the master can be used to monitor actual values.
   a) Click ¤Add Network Variable¤ in the lower part of the view.
   b) Select a producer object in selection combo box.
   c) Click ¤OK¤ to confirm selection.
   d) Repeat above steps for every network variable.

Network Variable from Slave to IEC 61131 Program



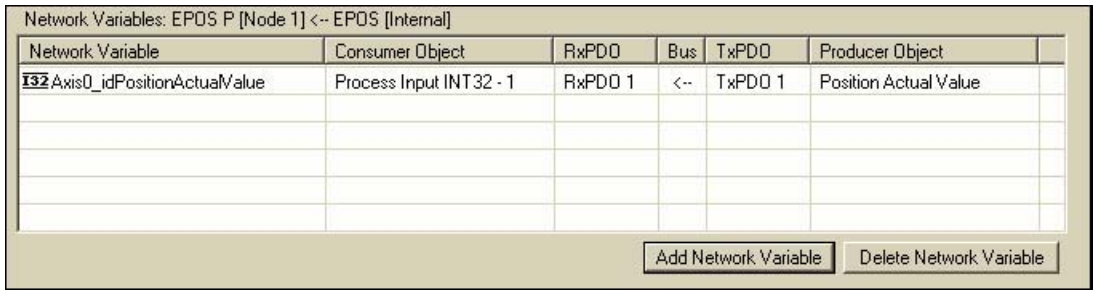


Figure 4-45 Input Network Variables

4) **Step 4: Network Variable File (*.poe)**
   a) Click browse button on the bottom of the view.
   b) Enter network variable file name for export and close dialog.

Figure 4-46 Network Variable File

5) **Step 5: Save Network Configuration and Export Network Variables**

   Click ¤OK¤ to save network configuration. The network variables are exported to selected network variable file.

6) **Step 6: Import Network Variables to IEC 61131 program**
   a) Open your IEC 61131 program in the programming tool «Open PCS».
   b) Select the menu item ¤Import¤ in the submenu “File” of the menu “File”.
   c) Click the context menu item “’Link to Active Resource’ to use the network variables.

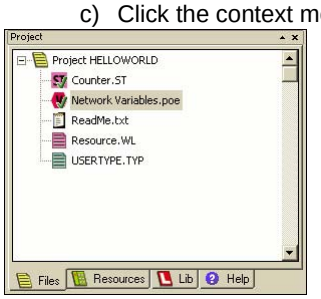


Figure 4-47 Project Browser in Programming Tool

Document ID: 786 912-07 V1 en
Edition: January 2010

5 Function Blocks

For every function block, you will find…

- a brief description,
- a block diagram,
- a table listing the available variables,
- remarks and explanations on the variables and their behavior, and
- the Function Block call in type.

Please observe below information prior engaging with functionalities of further describes function blocks.

Generally applicable Parameters

- *Function Block calls use programming language ST.*
- *Using the "Network Configuration Tool", axis number of internal and external axes may be set as desired. Thereby, respect permitted value range.*
- *The input/output variable **Axis** defines the addressed axis.*
- *The output variable **Error** signals an error having occurred during execution of the function block.*
- *The output variable **ErrorID** allows to get more information on the error cause.*
- *The output variable **Done** signals the successful read operation.*

Important! ***Generally applicable Rules***

The execution of a function block instance might take longer than one PLC cycle.

- *For a proper working system, a function block instance must be called (Execute or Enable) at every program cycle until its termination is signalled by the output **Done**, **Error** or **Abort**.*
- *Upon every call, the function block instance will continue at its actual internal state (at the position it stopped during the previous PLC program cycle). Breaking this rule will cause system errors, especially if the function block uses CAN communication services which might not have been finished fast enough.*

5.1 Motion Control Function Blocks

5.1.1 MC_Power

Controls the power stage of the axis (enabled or disabled).

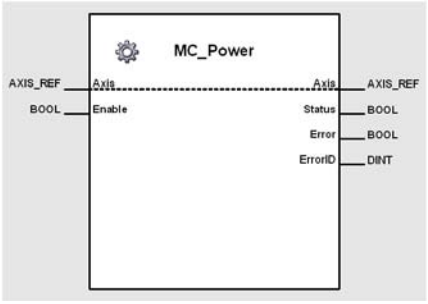


Figure 5-48 MC_Power

Important
MC_Power must be called until output "Status" has same value as input "Enable".

5.1.1.1 Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input/Output	Axis	AXIS_REF	0	0…31	AxisNo [USINT]
Input[*I)]	Enable	BOOL	FALSE	TRUE, FALSE	–
Output[*O)]	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-90	
	Status	BOOL	FALSE	TRUE, FALSE	–

I) As long as *Enable* is TRUE (positive state), the power stage of the axis is activated.
O) *Status* shows state of power stage.

Table 5-33 MC_Power – Variables

5.1.1.2 Call

```
------------------------------------------------------------------------------
(* 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);
```

5.1.2 MC_Reset

Resets all internal axis-related errors.

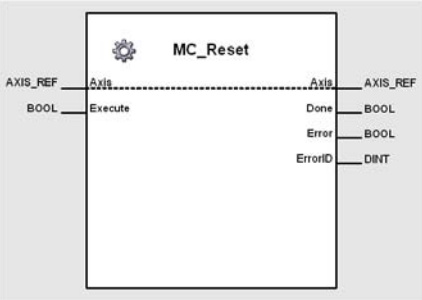


Figure 5-49 MC_Reset

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

5.1.2.1 Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input/Output	Axis	AXIS_REF	0	0…31	AxisNo [USINT]
Input*I)	Execute	BOOL	FALSE	TRUE, FALSE	–
Output*O)	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-90	–

I) At positive edge of *Execute*, axis status changes from Errorstop to StandStill. After execution of MC_Reset, the power stage must be re-enabled (➔"MC_Power" on page 5-42).

O) *Done* signals successful reset of axis status.

Table 5-34 MC_Reset – Variables

5.1.2.2 Call

```
-------------------------------------------------------------------------------
(* 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);
```

5.1.3 MC_ReadStatus

Returns the status of the axis with respect to the motion currently in progress.

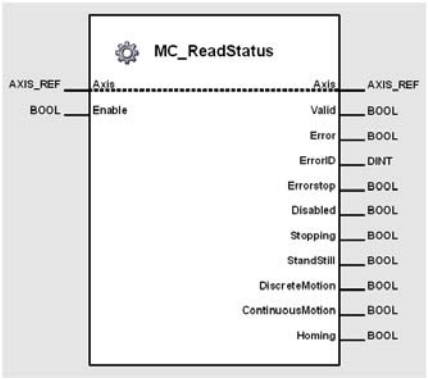


Figure 5-50 MC_ReadStatus

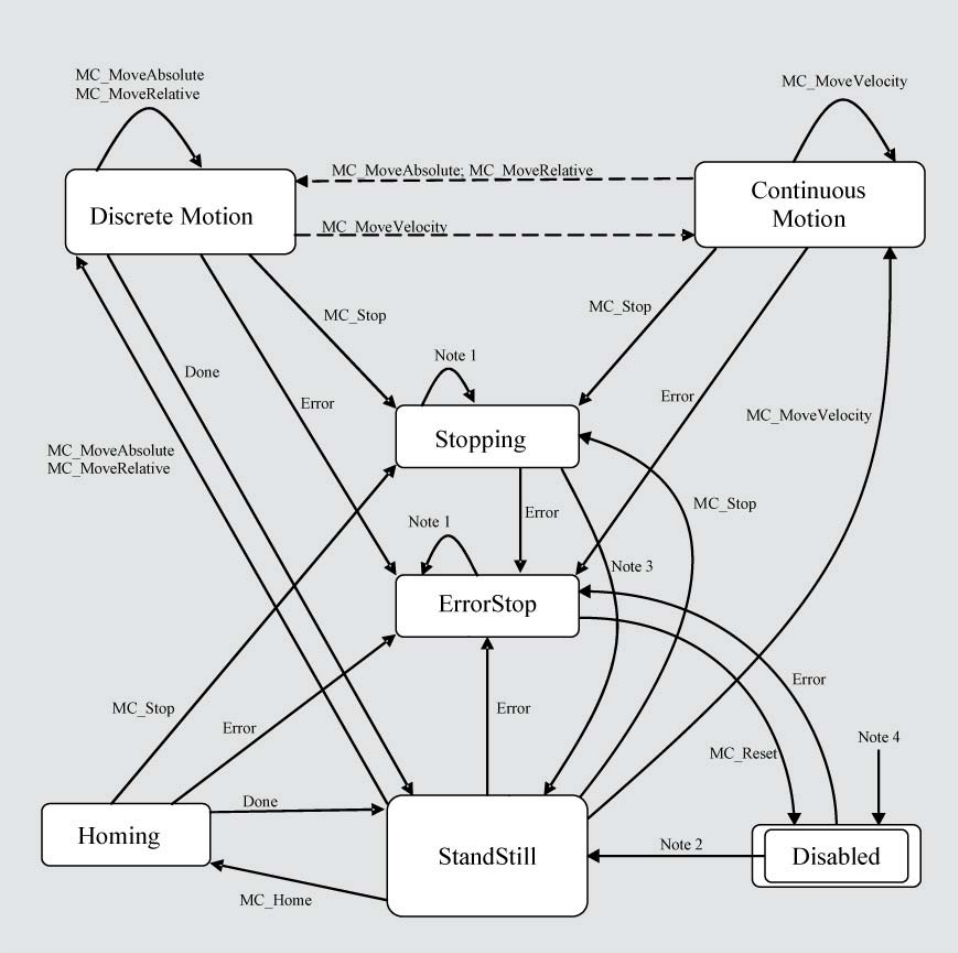
5.1.3.1 Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input/Output	Axis	AXIS_REF	0	0…31	AxisNo [USINT]
Input[*I)]	Enable	BOOL	FALSE	TRUE, FALSE	–
Output[*O)]	ContinuousMotion	BOOL	FALSE	TRUE, FALSE	–
	Disabled	BOOL	FALSE	TRUE, FALSE	–
	DiscreteMotio	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-90	–
	Errorstop	BOOL	FALSE	TRUE, FALSE	–
	Homing	BOOL	FALSE	TRUE, FALSE	–
	StandStill	BOOL	FALSE	TRUE, FALSE	–
	Stopping	BOOL	FALSE	TRUE, FALSE	–
	Valid	BOOL	FALSE	TRUE, FALSE	–

I) As long as *Enable* is TRUE (positive state), status parameter is continuously being read.
O) TRUE (positive state) of *Valid* signals successful update of axis status.

Table 5-35 MC_ReadStatus – Variables

Details on possible states (➔Figure 5-51).



Notes:

1) In *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.
2) Power.Enable = TRUE and no error present in the axis.
3) MC_Stop.Done
4) MC_Power.Enable = FALSE

Figure 5-51 MC_ReadStatus – States

5.1.3.2 Call

```
-------------------------------------------------------------------------------
(* 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;
```

5.1.4 MC_MoveAbsolute

Commands a controlled motion to a specified absolute position using a trapezoidal or sinusoidal profile.

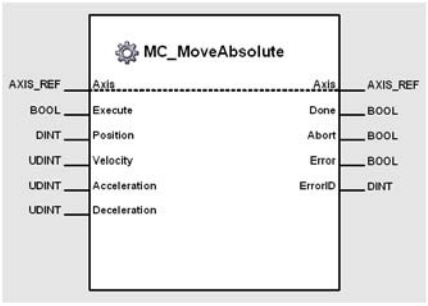


Figure 5-52 MC_MoveAbsolute

Important!

Execution of the instance might take longer than one PLC cycle (➔page 5-41).

5.1.4.1 Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input/Output	Axis	AXIS_REF	0	0…31	AxisNo [USINT]
Input[*I)]	Acceleration	UDINT	0	0…4‘294‘967’295	rpm/s
	Deceleration	UDINT	0	0…4‘294‘967’295	rpm/s
	Execute	BOOL	FALSE	TRUE, FALSE	–
	Position	DINT	0	-2‘147‘483‘648 … +2‘147‘483‘647	qc
	Velocity	UDINT	0	0…25‘000	rpm
Output[*O)]	Abort	BOOL	FALSE	TRUE, FALSE	–
	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-90	–

I) A positive edge of *Execute* triggers a new absolute movement using a profile corresponding to *Velocity*, *Acceleration* and *Deceleration*.
Position is defined in quad count (encoder increments) [qc].

O) Successful positioning is signalled with a positive value (TRUE) at *Done*. Execution of this instance is immediately stopped if another function block instance is executing movement using the same axis. In this case a positive state (TRUE) at *Abort* will be set.
Done, *Abort* and *Error* can be reset by a negative state (FALSE) to *Execute*. If *Execute* is reset before completion of positioning, *Done*, *Abort* and *Error* show status of positioning during one cycle, then they are reset to negative state (FALSE).
Velocity, *Acceleration* and *Deceleration* must only be defined upon first call – repeated calls will use value of first call and do not require further definition.

Table 5-36 MC_MoveAbsolute – Variables

Details on possible calling sequences (➔Figure 5-53).

- The first sequence shows two complete movements. The second instance will be initiated upon completion of the first movement.
- The second sequence shows an interrupted movement. Setting the variable Test will trigger the second instance while fist instance is being executed.

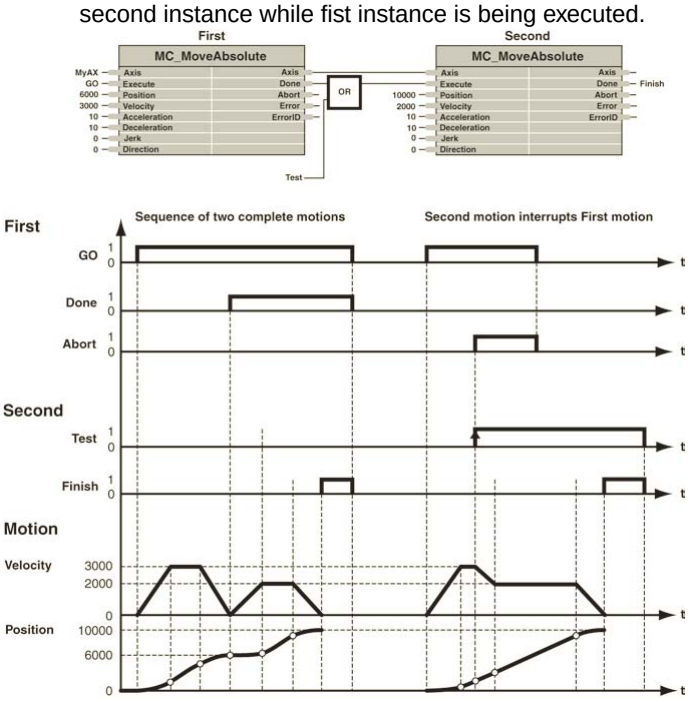


Figure 5-53 MC_MoveAbsolute – Sequence

5.1.4.2 Call

```
-------------------------------------------------------------------------------
(* 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,Decel-
eration:=50);
```

5.1.5 MC_MoveRelative

Commands a controlled motion of a specified distance relative to the actual position at the time of the execution using trapezoidal or sinusoidal profile. The new absolute target position is defined by the distance added to the actual position.

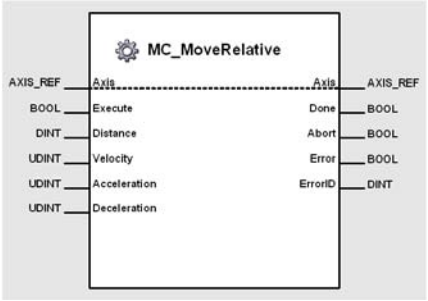


Figure 5-54 MC_MoveRelative

Important!
Execution of the instance might take longer than one PLC cycle (➔page 5-41).

5.1.5.1 Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input/Output	Axis	AXIS_REF	0	0…31	AxisNo [USINT]
Input[*I)]	Acceleration	UDINT	0	0…4‘294‘967’295	rpm/s
	Deceleration	UDINT	0	0…4‘294‘967’295	rpm/s
	Distance	DINT	0	-2‘147‘483‘648 … +2‘147‘483‘647	qc
	Execute	BOOL	FALSE	TRUE, FALSE	–
	Velocity	UDINT	0	0…25‘000	rpm
Output[*O)]	Abort	BOOL	FALSE	TRUE, FALSE	–
	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-90	–

I) A positive edge of *Execute* triggers a new absolute movement using a profile corresponding to *Velocity*, *Acceleration* and *Deceleration*. The defined distance is added to the actual position and commanded as a new absolute target position.
Distance is defined in quadcount (encoder increments) [qc].

O) Successful positioning is signalled with a positive value (TRUE) at *Done*. Execution of this instance is immediately stopped if another function block instance is executing movement using the same axis. In this case a positive state (TRUE) at *Abort* will be set.
Done, *Abort* and *Error* can be reset by a negative state (FALSE) to *Execute*. If *Execute* is reset before completion of positioning, *Done*, *Abort* and *Error* show status of positioning during one cycle, then they are reset to negative state (FALSE).
Velocity, *Acceleration* and *Deceleration* must only be defined upon first call – repeated calls will use value of first call and do not require further definition.

Table 5-37 MC_MoveRelative – Variables

Details on possible calling sequences (➔Figure 5-55).

- 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.

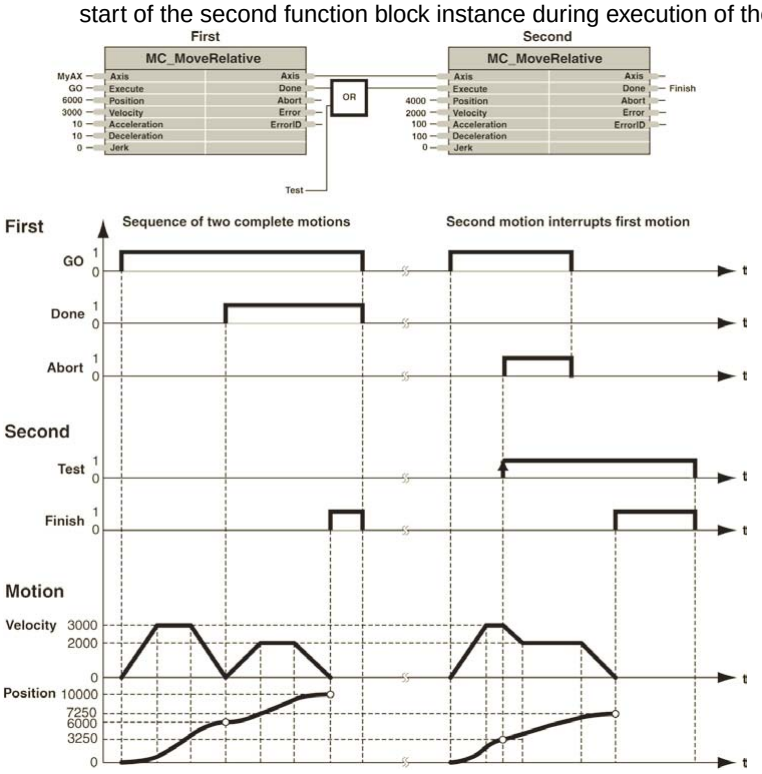


Figure 5-55 MC_MoveRelative – Sequence

5.1.5.2 Call

```
-------------------------------------------------------------------------------
(* 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);
```

5.1.6 MC_MoveVelocity

Commands a continuously controlled motion at a specified velocity using a trapezoidal or sinusoidal acceleration profile.

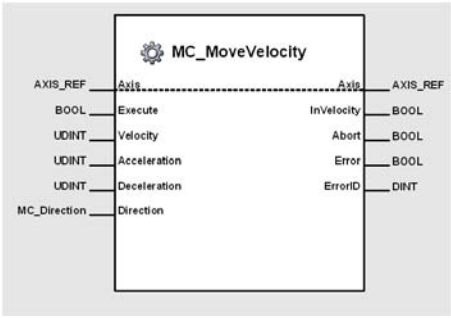


Figure 5-56 MC_MoveVelocity

Important!
Execution of the instance might take longer than one PLC cycle (➔page 5-41).

5.1.6.1 Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input/Output	Axis	AXIS_REF	0	0…31	AxisNo [USINT]
Input[*I)]	Acceleration	UDINT	0	0…4‘294‘967’295	rpm/s
	Deceleration	UDINT	0	0…4‘294‘967’295	rpm/s
	Execute	BOOL	FALSE	TRUE, FALSE	–
	Direction	Enum MC_Direction	MCposi-tive	MPpositive MPnegative	qc
	Velocity	UDINT	0	0…25‘000	rpm
Output[*O)]	Abort	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-90	–
	InVelocity	BOOL	FALSE	TRUE, FALSE	–

I) A positive edge of *Execute* triggers a new absolute continues velocity movement defined by *Velocity* using values of *Acceleration* and *Deceleration*.
MC_Stop will stop the movement. Another call changes the active velocity, thereby *Velocity* must be of positive value higher than 0.
Direction defines the movement direction and is defined in quadcount (encoder increments) [qc].

O) *InVelocity* signals achievement of commanded velocity. Another call executing a movement using the same axis will immediately *stop the movement. In this case a positive state (TRUE) at Abort will be set.*
InVelocity, Abort and Error can be reset by a negative state (FALSE) to *Execute*. If reset before completion of positioning, *InVelocity, Abort and Error* show status of positioning during one cycle, then they are reset to negative state (FALSE).
Velocity, *Acceleration* and *Deceleration* must only be defined upon first call – repeated calls will use value of first call and do not require further definition.

Table 5-38 MC_MoveVelocity – Variables

Details on possible calling sequences (➔Figure 5-57).

- 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.

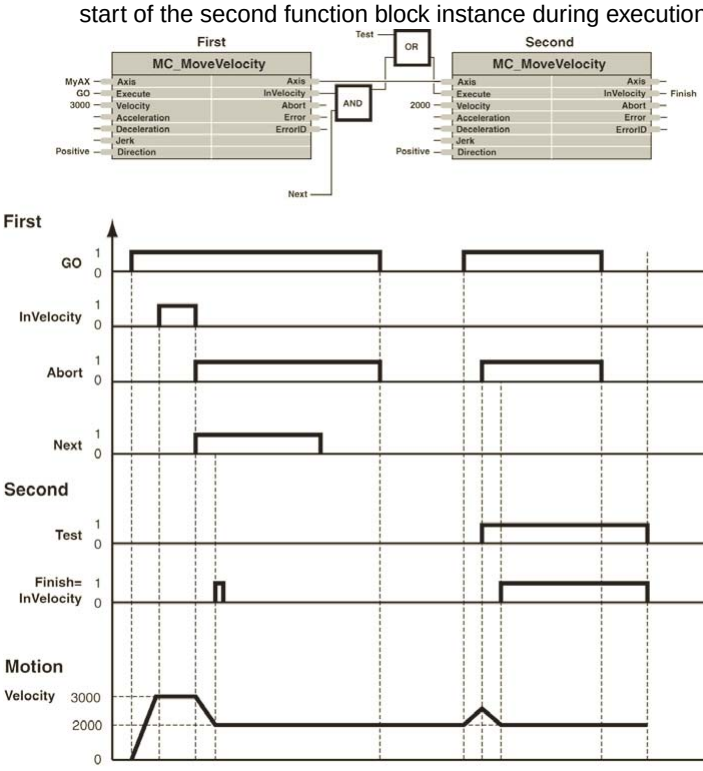


Figure 5-57 MC_MoveVelocity – Sequence

5.1.6.2 Call

```
-------------------------------------------------------------------------------
(* 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);
```

5.1.7 MC_Home

Commands the axis to perform the homing procedure. The absolute home position is determined using one of the available homing methods (for details ➔separate document «EPOS Firmware Specification»).

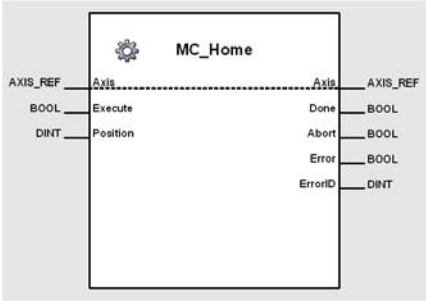


Figure 5-58 MC_Home

5.1.7.1 Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input/Output	Axis	AXIS_REF	0	0…31	AxisNo [USINT]
Input[*I)]	Execute	BOOL	FALSE	TRUE, FALSE	–
	Position	DINT	0	-2‘147‘483‘648 … +2‘147‘483‘647	qc
Output[*O)]	Abort	BOOL	FALSE	TRUE, FALSE	–
	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-90	–

I) A positive edge of *Execute* triggers a new homing procedure.
Position determines the new home position value after successful completion homing procedure and is defined in quadcount (encoder increments) [qc]. *Position* must only be defined upon first call – repeated calls will use value of first call and do not require further definition.
Additional parameters for a homing procedure must be configured using MC_WriteParameter (➔page 5-58), for detailed information ➔separate document «EPOS Firmware Specification».

O) *Done* signals successful termination of the procedure.
If another instance is starting a homing procedure using the same axis, the execution of the first instance is immediately being stopped, Abort is set to positive state (TRUE).
Done, *Abort* and *Error* can be reset by a negative state (FALSE) to *Execute*. If *Execute* is reset before completion of positioning, *Done*, *Abort* and *Error* show status of positioning during one cycle, then they are reset to negative state (FALSE).

Table 5-39 MC_Home – Variables

5.1.7.2 Call

```
-------------------------------------------------------------------------------
(* 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);
```

5.1.8 MC_Stop

Commands a controlled motion stop of the axis using a trapezoidal or sinusoidal deceleration profile.

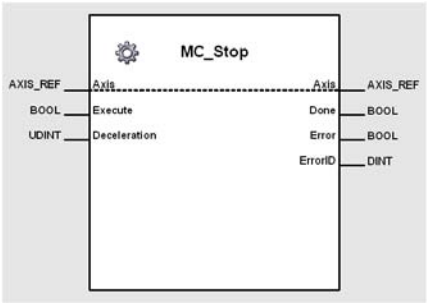


Figure 5-59 MC_Stop

5.1.8.1 Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input/Output	Axis	AXIS_REF	0	0…31	AxisNo [USINT]
Input[*I)]	Execute	BOOL	FALSE	TRUE, FALSE	–
	Deceleration	UDINT	0	0…4'294'967'295	rpm/s
Output[*O)]	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-90	–
	InVelocity	BOOL	FALSE	TRUE, FALSE	–

I) A positive edge of *Execute* stops the axis using a defined deceleration profile.

O) *Done* and *Error* are reset by setting a negative state (FALSE) to *Execute*. If *Execute* is reset before completion of positioning, *Done* and *Error* will continue to signal the stoppage during one cycle, and are then reset to negative state (FALSE).

Table 5-40 MC_Stop – Variables

5.1.8.2 Call

```
------------------------------------------------------------------------------
(* 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);
```

5.1.9 MC_ReadParameter

Returns an axis parameter value.

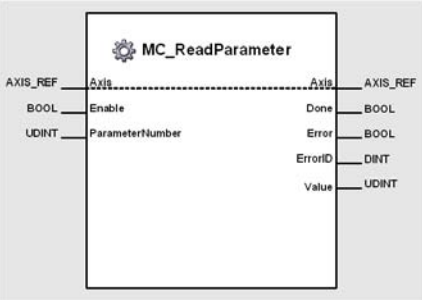


Figure 5-60 MC_ReadParameter

Important!
Execution of the instance might take longer than one PLC cycle (➔page 5-41).

5.1.9.1 Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input/Output	Axis	AXIS_REF	0	0…31	AxisNo [USINT]
Input*I)	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)	
Output*O)	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-90	–
	Value	UDINT	0	0…4'294'967'295	–

I) As long as *Enable* is TRUE (positive state), the value of a specified parameter will continuously be read.
ParameterNumber is defining the parameter to be read. Besides the listed parameter, CANopen objects can be read using *ParameterNumber* as a multiplexer. Thus, allowing to read all EPOS objects from the object dictionary (➔separate document «EPOS P Firmware Specification»).
The multiplexer (for details ➔"Multiplexer Example" on page 5-56) 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).

O) *Value* allows retrieval of the value.

Table 5-41 MC_ReadParameter – Variables

5.1.9.2 Multiplexer Example

ParameterNumber.....= 16#207C0102
Name.....= Analog Input 1 of EPOS
Object Index.....= 16#207C
Object Sub-index.....= 16#01
Object Length.....= 16#02

5.1.9.3 Call

```
-------------------------------------------------------------------------------
(* 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);
```

5.1.10 MC_ReadBoolParameter

Returns an axis parameter value.

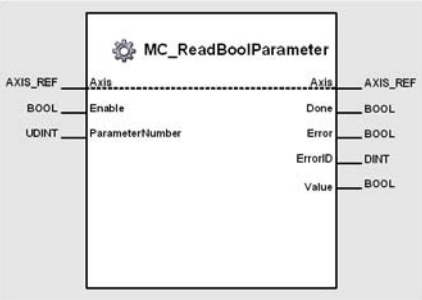


Figure 5-61 MC_ReadBoolParameter

Important!
Execution of the instance might take longer than one PLC cycle (➔page 5-41).

5.1.10.1 Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input/Output	Axis	AXIS_REF	0	0…31	AxisNo [USINT]
Input*I)	Enable	BOOL	FALSE	TRUE, FALSE	–
	ParameterNumber	UDINT	0	4 EnableLimitPos 5 EnableLimitNeg 6 EnablePosLagMonitoring	
Output*O)	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-90	–
	Value	UDINT	0	0…4'294'967'295	–

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

O) *Value* allows retrieval of the value.

Table 5-42 MC_ReadBoolParameter – Variables

5.1.10.2 Call

```
------------------------------------------------------------------------------
(* 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);
```

5.1.11 MC_WriteParameter

Modifies the value of an axis parameter.

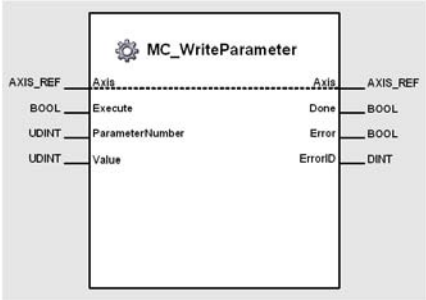


Figure 5-62 MC_WriteParameter

Important!
Execution of the instance might take longer than one PLC cycle (➔page 5-41).

5.1.11.1 Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input/Output	Axis	AXIS_REF	0	0…31	AxisNo [USINT]
Input[*I)]	Enable	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…4'294'967'295	–
Output[*O)]	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-90	–

I) A positive edge at *Execute* triggers a write operation of the specified parameter. *ParameterNumber* is defining the parameter to be written. Besides the listed parameter, CANopen objects can be read using *ParameterNumber* as a multiplexer. Thus, allowing to read all EPOS objects from the object dictionary (➔separate document «EPOS P Firmware Specification»).
The multiplexer (for details ➔"Multiplexer Example" on page 5-59) 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).

O) Successful write operation is signalled with a positive value (TRUE) at *Done*.

Table 5-43 MC_WriteParameter – Variables

5.1.11.2 Multiplexer Example

ParameterNumber ..... = 16#20780102
Name ..... = Digital Outputs of EPOS
Object Index..... = 16#2078
Object Sub-index ..... = 16#01
Object Length ..... = 16#02

5.1.11.3 Call

```
---------------------------------------------------------------------------------
(* 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);
```

5.1.12 MC_ReadActualPosition

Returns the actual position of an axis.

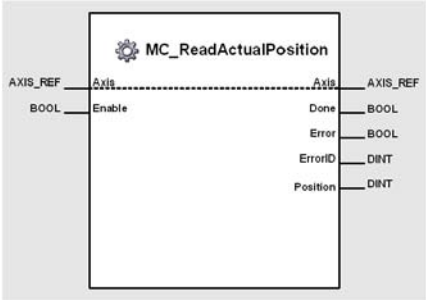


Figure 5-63 MC_ReadActualPosition

Important!
Execution of the instance might take longer than one PLC cycle (➔page 5-41).

5.1.12.1 Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input/Output	Axis	AXIS_REF	0	0…31	AxisNo [USINT]
Input*I)	Enable	BOOL	FALSE	TRUE, FALSE	–
Output*O)	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-90	–
	Position	DINT	0	-2’147’483’648 [min(DINT)] … +2’147’483’647 [max(DINT)]	qc

I) As long as *Enable* is TRUE (positive state), the actual position will continuously be read.
O) The actual position can be retrieved from *Position*.
Position is defined in quadcount (encoder increments) [qc].

Table 5-44 MC_ReadActualPosition – Variables

5.1.12.2 Call

```
-------------------------------------------------------------------------------
(* 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);
```

5.1.13 MC_ReadActualVelocity

Returns the actual velocity of an axis.

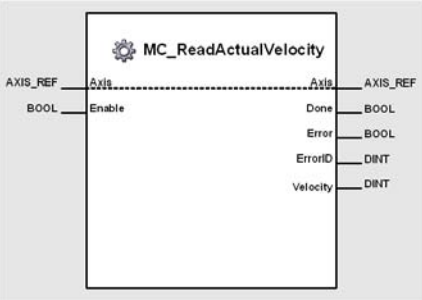


Figure 5-64 MC_ReadActualVelocity

Important!
Execution of the instance might take longer than one PLC cycle (➔page 5-41).

5.1.13.1 Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input/Output	Axis	AXIS_REF	0	0…31	AxisNo [USINT]
Input[*I)]	Enable	BOOL	FALSE	TRUE, FALSE	–
Output[*O)]	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-90	–
	Velocity	DINT	0	-2'147'483'648 [min(DINT)] … +2'147'483'647 [max(DINT)]	rpm

I) As long as *Enable* is TRUE (positive state), the actual velocity will continuously be read.
O) The actual velocity can be retrieved from *Velocity*.

Table 5-45 MC_ReadActualVelocity – Variables

5.1.13.2 Call

```
-------------------------------------------------------------------------------
(* 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);
```

5.1.14 MC_ReadActualCurrent

Returns the actual current of an axis.

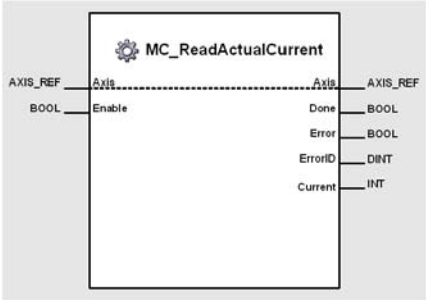


Figure 5-65 MC_ReadActualCurrent

Important!
Execution of the instance might take longer than one PLC cycle (➔page 5-41).

5.1.14.1 Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input/Output	Axis	AXIS_REF	0	0…31	AxisNo [USINT]
Input*I)	Enable	BOOL	FALSE	TRUE, FALSE	–
Output*O)	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-90	–
	Current	INT	0	- 32768 [min(INT)] … + 32767 [max(INT)]	mA

I) As long as *Enable* is TRUE (positive state), the actual current will continuously be read.
O) The actual velocity can be retrieved from *Current*.

Table 5-46 MC_ReadActualCurrent – Variables

5.1.14.2 Call

```
-------------------------------------------------------------------------------
(* 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);
```

5.1.15 MC_ReadAxisError

Returns the first entry in the error history of the EPOS device.

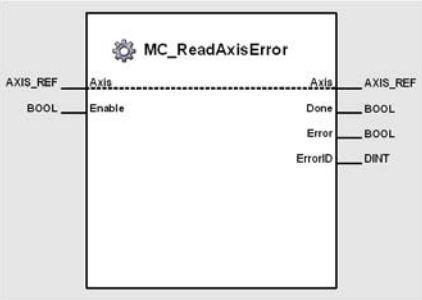


Figure 5-66 MC_ReadAxisError

5.1.15.1 Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input/Output	Axis	AXIS_REF	0	0…31	AxisNo [USINT]
Input[*I)]	Enable	BOOL	FALSE	TRUE, FALSE	–
Output[*O)]	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	EPOS device error ➔item “[7]” on page 1-8	–

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

O) With successful operation (*Error* = FALSE), *ErrorID* contains the axis error (➔item “[7]” on page 1-8).

Table 5-47 MC_ReadAxisError – Variables

5.2 Maxon Utility Function Blocks

5.2.1 MU_GetAllDigitalInputs

Returns the state of all digital inputs.

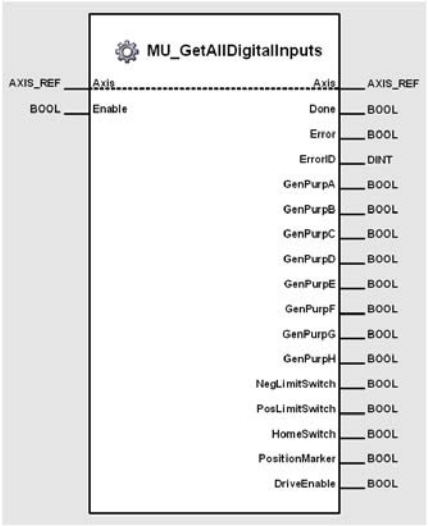


Figure 5-67 MU_GetAllDigitalInputs

5.2.1.1 Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input/Output	Axis	AXIS_REF	0	0…31	AxisNo [USINT]
Input[*I)]	Enable	BOOL	FALSE	TRUE, FALSE	–
Output	Done	BOOL	FALSE	TRUE, FALSE	–
	Drive Enable	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-90	–
	GenPurpA	BOOL	FALSE	TRUE, FALSE	–
	GenPurpB	BOOL	FALSE	TRUE, FALSE	–
	GenPurpB	BOOL	FALSE	TRUE, FALSE	–
	GenPurpD	BOOL	FALSE	TRUE, FALSE	–
	GenPurpE	BOOL	FALSE	TRUE, FALSE	–
	GenPurpF	BOOL	FALSE	TRUE, FALSE	–
	GenPurpG	BOOL	FALSE	TRUE, FALSE	–
	GenPurpH	BOOL	FALSE	TRUE, FALSE	–
	HomeSwitch	BOOL	FALSE	TRUE, FALSE	–
	NegLimitSwitch	BOOL	FALSE	TRUE, FALSE	–
	PositionMarker	BOOL	FALSE	TRUE, FALSE	–
	PosLimitSwitch	BOOL	FALSE	TRUE, FALSE	–

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

Table 5-48 MU_GetAllDigitalInputs – Variables

5.2.1.2 Call

```
-------------------------------------------------------------------------------
(* 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);
```

5.2.2 MU_GetDigitalInput

Returns the state of a specific digital input.

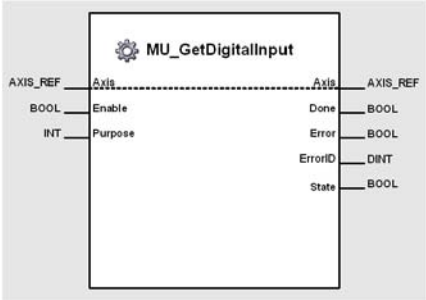


Figure 5-68 MU_GetDigitalInput

5.2.2.1 Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input/Output	Axis	AXIS_REF	0	0…31	AxisNo [USINT]
Input[*I)]	Enable	BOOL	FALSE	TRUE, FALSE	–
	Purpose	INT	0	NegLimitSwitch = 0, PosLimitSwitch = 1, HomeSwitch = 2, PositionMarker = 3, Enable = 4, GenPurpH = 8, GenPurpG = 9, GenPurpF = 10, GenPurpE = 11, GenPurpD = 12, GenPurpC = 13, GenPurpB = 14, GenPurpA = 15	–
Output	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-90	–
	State	BOOL	FALSE	TRUE, FALSE	–

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

Table 5-49 MU_GetDigitalInput – Variables

5.2.2.2 Call

```
------------------------------------------------------------------------------
(* 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);
```

5.2.3 MU_GetAnalogInput

Returns the state of a specific digital input.

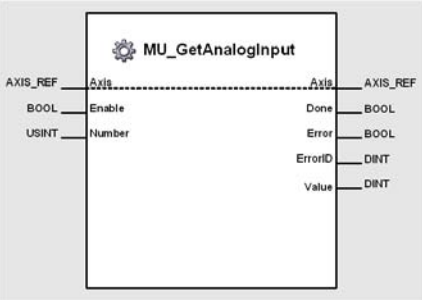


Figure 5-69 MU_GetAnalogInput

5.2.3.1 Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input/Output	Axis	AXIS_REF	0	0…31	AxisNo [USINT]
Input[*I)]	Enable	BOOL	FALSE	TRUE, FALSE	–
	Number	USINT	0	1, 2	–
Output	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-90	–
	Value	DINT	0	0…5'000	vV

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

Table 5-50 MU_GetAnalogInput – Variables

5.2.3.2 Call

```
-------------------------------------------------------------------------------
(* 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);
```

5.2.4 MU_SetAllDigitalOutputs

Modifies the value of all digital outputs.

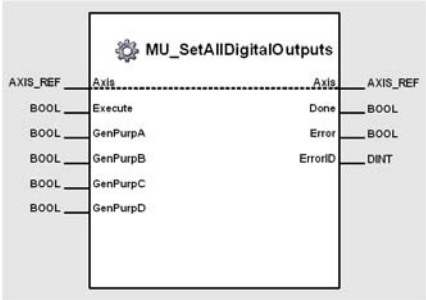


Figure 5-70 MU_SetAllDigitalOutputs

5.2.4.1 Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input/Output	Axis	AXIS_REF	0	0…31	AxisNo [USINT]
Input[*I)]	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	–
Output	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-90	–

I) A positive edge of *Execute* triggers a write operation of all digital outputs.

Table 5-51 MU_SetAllDigitalOutputs – Variables

5.2.4.2 Call

```
-------------------------------------------------------------------------------
(* 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);
```

5.2.5 MU_GetDeviceErrorCount

Returns the number of actual errors.

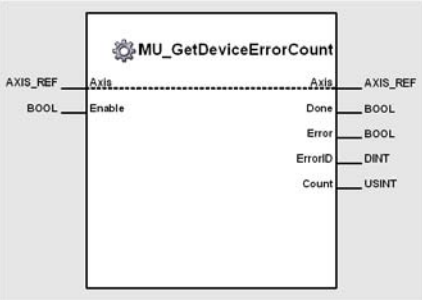


Figure 5-71 MU_GetDeviceErrorCount

5.2.5.1 Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input/Output	Axis	AXIS_REF	0	0…31	AxisNo [USINT]
Input*I)	Enable	BOOL	FALSE	TRUE, FALSE	–
Output*O)	Count	USINT	0	0…255	–
	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-90	–

I) As long as *Enable* is TRUE (positive state), the number of existing errors will continuously be read.

O) The actual number of existing errors can be read from *Count*.

Table 5-52 MU_GetDeviceErrorCount – Variables

5.2.5.2 Call

```
------------------------------------------------------------------------------------
(* 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);
```

5.2.6 MU_GetDeviceError

Returns the error code of a specific entry in the error history.

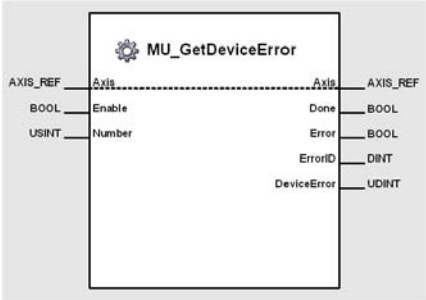


Figure 5-72 MU_GetDeviceError

5.2.6.1 Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input/Output	Axis	AXIS_REF	0	0…31	AxisNo [USINT]
Input[*I)]	Enable	BOOL	FALSE	TRUE, FALSE	–
	Number	USINT	1	1…count (➔page 5-69)	–
Output[*O)]	DeviceError	UDINT	0	0…4'294'967'265	–
	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-90	–

I) As long as *Enable* is TRUE (positive state), the error code of a specific entry in the error history will continuously be read.

O) The error code can be read from *DeviceError*.

Table 5-53 MU_GetDeviceError – Variables

5.2.6.2 Call

```
------------------------------------------------------------------------------
(* 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 his-
tory *)
fbGetDeviceErrorCount(Axis := myAxis, Enable := TRUE, Number := 2);
```

5.2.7 MU_GetObject

Returns the value of an EPOS object.

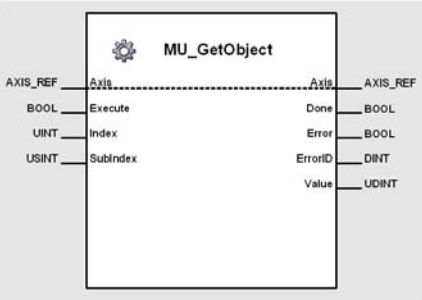


Figure 5-73 MU_GetObject

5.2.7.1 Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input/Output	Axis	AXIS_REF	0	0…31	AxisNo [USINT]
Input[*I)]	Execute	BOOL	FALSE	TRUE, FALSE	–
	Index	UINT	0	0…65'535	–
	SubIndex	USINT	0	0…255	–
Output[*O)]	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-90	–
	Value	UDINT	0	0…4'294'967'265	–

I) A positive edge of *Execute* triggers a read operation of a specific EPOS object. *Index* and *SubIndex* define the object to be read.

O) The value of the object can be read from the *Value*.

Table 5-54 MU_GetObject – Variables

5.2.7.2 Call

```
------------------------------------------------------------------------------
(* 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);
```

5.2.8 MU_SetObject

Modifies the value of an EPOS object.

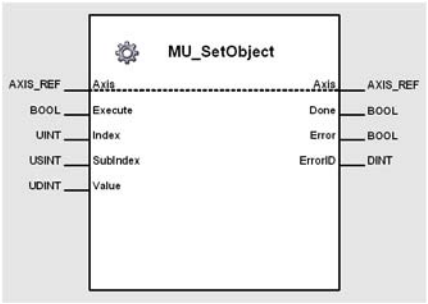


Figure 5-74 MU_SetObject

5.2.8.1 Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input/Output	Axis	AXIS_REF	0	0…31	AxisNo [USINT]
Input[*I)]	Execute	BOOL	FALSE	TRUE, FALSE	–
	Index	UINT	0	0…65'535	–
	SubIndex	USINT	0	0…255	–
	Value	UDINT	0	0…4'294'967'265	–
Output	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-90	–

I) A positive edge of *Execute* triggers a write operation of a specific EPOS object. *Index* and *SubIndex* define the object to be modified.

Table 5-55 MU_SetObject – Variables

5.2.8.2 Call

```
-------------------------------------------------------------------------------
(* 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);
```

5.2.9 MU_GetHomingParameter

Returns the values of the EPOS homing objects.

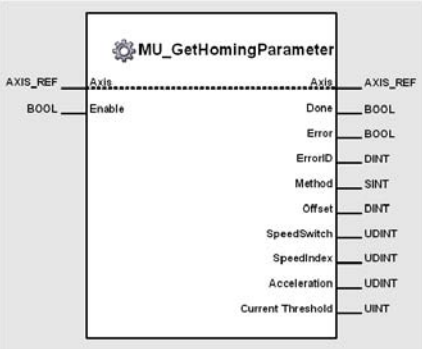


Figure 5-75 MU_GetHomingParameter

5.2.9.1 Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input/Output	Axis	AXIS_REF	0	0…31	AxisNo [USINT]
Input*I)	Enable	BOOL	FALSE	TRUE, FALSE	–
Output*O)	Acceleration	UDINT	1000	0…4294967295	rpm/s
	CurrentThreshold	UINT	500	0 and up (depending on hardware)	mA
	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-90	–
	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	-2‘147‘483‘648 … +2‘147‘483‘647	qc
	SpeedIndex	UDINT	10	0…4294967295	rpm
	SpeedSwitch	UDINT	100	0…4294967295	rpm

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

O) The values of the objects can be read from *Method*, *Offset*, *SpeedSwitch*, *SpeedIndex*, *Acceleration* and *CurrentThreshold*.

Table 5-56 MU_GetHomingParameter – Variables

5.2.9.2 Call

```
-------------------------------------------------------------------------------
(* 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);
```

5.2.10 MU_SetHomingParameter

Modifies the values of the EPOS homing objects.

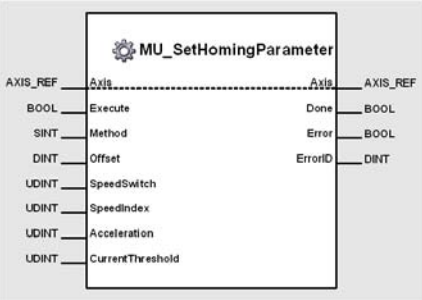


Figure 5-76 MU_SetHomingParameter

5.2.10.1 Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input/Output	Axis	AXIS_REF	0	0…31	AxisNo [USINT]
Input[*I)]	Acceleration	UDINT	1000	0…4294967295	rpm/s
	CurrentThreshold	UINT	500	0 and up (depending on hardware)	mA
	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	-2‘147‘483‘648 … +2‘147‘483‘647	qc
	SpeedIndex	UDINT	10	0…4’294’967’295	rpm
	SpeedSwitch	UDINT	100	0…4’294’967’295	rpm
Output	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-90	–

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

Table 5-57 MU_SetHomingParameter – Variables

5.2.10.2 Call

```
-------------------------------------------------------------------------------
(* 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);
```

5.2.11 MU_Selection

Selects between two values.

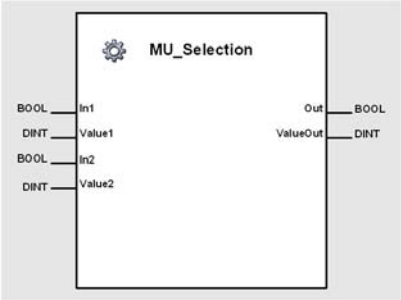


Figure 5-77 MU_Selection

5.2.11.1 Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input[*I)]	In1	BOOL	FALSE	TRUE, FALSE	–
	Value1	DINT	0	-2‘147‘483‘648 … +2‘147‘483‘647	qc
	In2	BOOL	FALSE	TRUE, FALSE	–
	Value2	DINT	0	-2‘147‘483‘648 … +2‘147‘483‘647	qc
Output[*O)]	Out	BOOL	FALSE	TRUE, FALSE	–
	ValueOut	DINT	0	-2‘147‘483‘648 … +2‘147‘483‘647	qc

I) *In1* selects *Value1*, *In2* selects *Value2*. If *In1* and *In2* are TRUE, *In1* is prioritized.
O) *Out* indicates a valid value of *ValueOut*.

Table 5-58 MU_Selection – Variables

5.2.11.2 Call

```
-------------------------------------------------------------------------------
(* 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);
```

5.2.12 MU_GetBitState

Extracts the state of a specific bit.

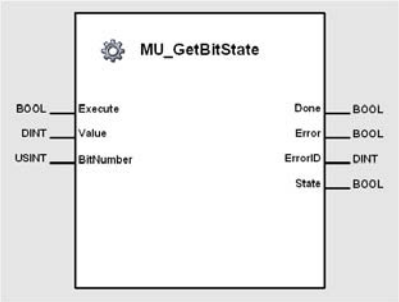


Figure 5-78 MU_GetBitState

5.2.12.1 Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input[*I)]	BitNumber	USINT	0	0…255	–
	Execute	BOOL	FALSE	TRUE, FALSE	–
	Value	DINT	0	-2‘147‘483‘648 … +2‘147‘483‘647	qc
Output	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-90	–
	State	BOOL	FALSE	TRUE, FALSE	–

I) A positive edge of *Execute* triggers a read operation of the state of a specific bit within *Value*. *BitNumber* defines the bit to be read.

Table 5-59 MU_GetBitState – Variables

5.2.12.2 Call

```
-------------------------------------------------------------------------------
(* 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*)
```

5.2.13 MU_SetBitState

Modifies the state of a specific bit within a given value.

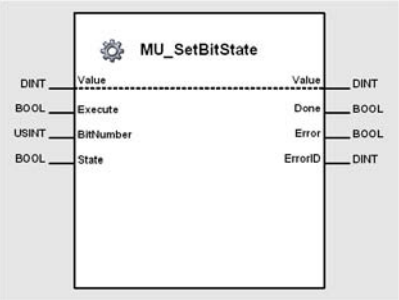


Figure 5-79 MU_SetBitState

5.2.13.1 Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input/Output	Axis	AXIS_REF	0	0…31	AxisNo [USINT]
Input[*I)]	BitNumber	USINT	0	0…255	–
	Execute	BOOL	FALSE	TRUE, FALSE	–
	State	BOOL	FALSE	TRUE, FALSE	–
Output	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-90	–

I) A positive edge of *Execute* triggers a read operation of the state of a specific bit within *Value*. *BitNumber* defines the bit to be written with the value in *State*.

Table 5-60 MU_SetBitState – Variables

5.2.13.2 Call

```
------------------------------------------------------------------------------
(* 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*)
```

5.3 CANopen DS-301 Function Blocks

5.3.1 CAN_Nmt

Permits change of network management state of a CANopen device.

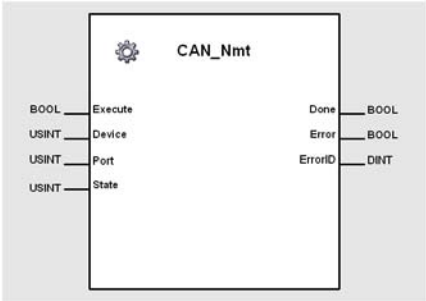


Figure 5-80 CAN_Nmt

5.3.1.1 Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input[*I)]	Device	USINT	0	0…127	–
	Execute	BOOL	FALSE	TRUE, FALSE	–
	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	
Output	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-90	–

I) A positive edge of *Execute* triggers the NMT service operation. The network management state of the defined device is changed.
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 *Port*.
Port distinguishes between internal and external CAN network.
State is define by CANopen (➔CANopen specification).

Table 5-61 CAN_Nmt – Variables

5.3.1.2 Call

```
-------------------------------------------------------------------------------
(* 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);
```

5.3.2 CAN_SdoRead

Permits reading of a CANopen object using the SDO protocol.

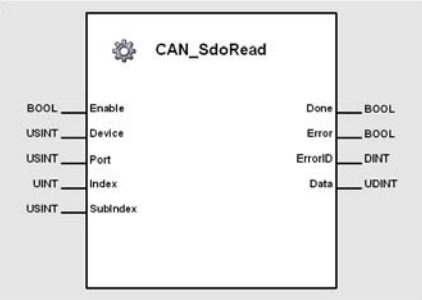


Figure 5-81 CAN_SdoRead

Important!
Execution of the instance might take longer than one PLC cycle (➔page 5-41).

5.3.2.1 Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input[*I)]	Device	USINT	0	0…127	–
	Enable	BOOL	FALSE	TRUE, FALSE	–
	Port	USINT	0	0 = internal port 1 = external port	–
	Index	UINT	0	UINT	–
	SubIndex	USINT	0	USINT	–
Output	Data	UDINT	0	0…4'294'967'295	–
	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-90	–

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

Table 5-62 CAN_SdoRead – Variables

5.3.2.2 Call

```
-------------------------------------------------------------------------------
(* 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);
```

5.3.3 CAN_SdoWrite

Permits writing of a CANopen object using the SDO protocol.

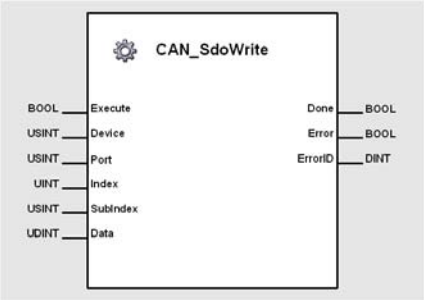


Figure 5-82 CAN_SdoWrite

Important!
Execution of the instance might take longer than one PLC cycle (➔page 5-41).

5.3.3.1 Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input[*I)]	Device	USINT	0	0…127	–
	Execute	BOOL	FALSE	TRUE, FALSE	–
	Port	USINT	0	0 = internal port 1 = external port	–
	Index	UINT	0	UINT	–
	SubIndex	USINT	0	USINT	–
	Data	UDINT	0	UDINT	–
Output	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-90	–

I) A positive edge of *Execute* triggers the write operation of a CANopen object.
The object is specified by *Index* and *SubIndex*.
Device corresponds to the CAN Node-ID.
Port distinguishes between internal and external CAN network.

Table 5-63 CAN_SdoWrite – Variables

5.3.3.2 Call

```
-------------------------------------------------------------------------------
(* 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);
```

6 Markers

Markers are typically used to build intermediate results. They will be buffered in the PLC and do not have direct influence to the outputs. By using markers, extensive operations can be essentially simplified. Further, they act as transmitter between different modules.

EPOS P and MCD EPOS P are using specific marker areas for error and warning information.

6.1 User Marker Area

Length is 25 entries (32-bit values), write or read are supported. 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;

Table 6-64 User Marker Variables – Examples

6.2 Marker Global Status Register

Length is 25 entries (32-bit values), write or read are supported. It holds the EPOS P global status register and is identical to the EPOS P CANopen object 0x1002.

Bit 0 to Bit 7 represent an overview of the CANopen slave error registers. If a connected CANopen slaves reports an error register flag, the according bit will be set. For the meaning of CANopen error register, please refer to the connected CANopen slave's object description (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 6-65 Global Status Register Markers

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;

Table 6-66 Global Status Register Markers – Examples

6.3 Marker Global Axis Error Register

Length is 32-bit. It holds the EPOS P global status register and is identical to 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	Axis 0 is in error state
1	Axis 1 is in error state
2	Axis 2 is in error state

Bit	Description
n	Axis n is in error state
31	Axis 31 is in error state

Table 6-67 Global Axis Error Register Markers

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;

Table 6-68 Global Axis Error Register Markers – Examples

6.4 Reserved Marker Area

Length is 23 entries (32-bit values). Reserved for future use.

6.5 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;

Table 6-69 CANopen Slave Error Register Markers – Examples 1

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;

Table 6-70 CANopen Slave Error Register Markers – Examples 2

7 Process I/Os

Process inputs and outputs are used to read incoming or write outgoing CANopen PDOs. Nevertheless, before this communication method can be used, PDO configuration will be required. For details ➔chapter “4.3 Network Configuration” on page 4-25.

Best Practice

- *Use PDO communication for powerful and easy data exchange to read/write direct addressed variables.*
- *Use the “Network Configuration Tool” to setup PDO communication and to employ Functional Blocks (➔chapter “5 Function Blocks” on page 5-41).*

The “Process Inputs Area” provides 6 declaration types of 16 values with a length of 8-bit each. The variable holds the same value as the related CANopen object dictionary entry with its respective index.

The following tables list the editable items that compose to the PDOs. Each table comes along with an example showing a variable containing colored marks on where to find respective elements and content.

7.1 Process Inputs

Example: InVar**1** AT %**IB0.0** : **SINT**; (*Object Index **0xA000**, Subindex **1***)

InVar/ Subindex	SINT		USINT		INT		UINT		DINT		UDINT	
	Decl.	Index	Decl.	Index	Decl.	Index	Decl.	Index	Decl.	Index	Decl.	Index
1	IB0.0	0xA000	IB16.0	0xA040	IW32.0	0xA0C0	IW64.0	0xA100	ID96.0	0xA1C0	ID160.0	0xA200
2	IB1.0	0xA000	IB17.0	0xA040	IW34.0	0xA0C0	IW66.0	0xA100	ID100.0	0xA1C0	ID164.0	0xA200
3	IB2.0	0xA000	IB18.0	0xA040	IW36.0	0xA0C0	IW68.0	0xA100	ID104.0	0xA1C0	ID168.0	0xA200
4	IB3.0	0xA000	IB19.0	0xA040	IW38.0	0xA0C0	IW70.0	0xA100	ID108.0	0xA1C0	ID172.0	0xA200
5	IB4.0	0xA000	IB20.0	0xA040	IW40.0	0xA0C0	IW72.0	0xA100	ID112.0	0xA1C0	ID174.0	0xA200
6	IB5.0	0xA000	IB21.0	0xA040	IW42.0	0xA0C0	IW74.0	0xA100	ID116.0	0xA1C0	ID180.0	0xA200
7	IB6.0	0xA000	IB22.0	0xA040	IW44.0	0xA0C0	IW76.0	0xA100	ID120.0	0xA1C0	ID184.0	0xA200
8	IB7.0	0xA000	IB23.0	0xA040	IW46.0	0xA0C0	IW78.0	0xA100	ID124.0	0xA1C0	ID188.0	0xA200
9	IB8.0	0xA000	IB24.0	0xA040	IW48.0	0xA0C0	IW80.0	0xA100	ID128.0	0xA1C0	ID192.0	0xA200
10	IB9.0	0xA000	IB25.0	0xA040	IW50.0	0xA0C0	IW82.0	0xA100	ID132.0	0xA1C0	ID196.0	0xA200
11	IB10.0	0xA000	IB26.0	0xA040	IW52.0	0xA0C0	IW84.0	0xA100	ID136.0	0xA1C0	ID200.0	0xA200
12	IB11.0	0xA000	IB27.0	0xA040	IW54.0	0xA0C0	IW86.0	0xA100	ID140.0	0xA1C0	ID204.0	0xA200
13	IB12.0	0xA000	IB28.0	0xA040	IW56.0	0xA0C0	IW88.0	0xA100	ID144.0	0xA1C0	ID208.0	0xA200
14	IB13.0	0xA000	IB29.0	0xA040	IW58.0	0xA0C0	IW90.0	0xA100	ID148.0	0xA1C0	ID212.0	0xA200
15	IB14.0	0xA000	IB31.0	0xA040	IW60.0	0xA0C0	IW92.0	0xA100	ID152.0	0xA1C0	ID216.0	0xA200
16	IB15.0	0xA000	IB31.0	0xA040	IW62.0	0xA0C0	IW94.0	0xA100	ID156.0	0xA1C0	ID220.0	0xA200

Table 7-71 Process Inputs – IEC 61131 Declaration Examples with Variables

7.2 Process Outputs

Example: OutVar16 AT %QD220.0 : UDINT; (*Object Index 0xA680, Subindex 16*)

OutVar/ Subindex	SINT		USINT		INT		UINT		DINT		UDINT	
	Decl.	Index	Decl.	Index	Decl.	Index	Decl.	Index	Decl.	Index	Decl.	Index
1	QB0.0	0xA480	QB16.0	0xA4C0	QW32.0	0xA540	QW64.0	0xA580	QD96.0	0xA640	QD160.0	0xA680
2	QB1.0	0xA480	QB17.0	0xA4C0	QW34.0	0xA540	QW66.0	0xA580	QD100.0	0xA640	QD164.0	0xA680
3	QB2.0	0xA480	QB18.0	0xA4C0	QW36.0	0xA540	QW68.0	0xA580	QD104.0	0xA640	QD168.0	0xA680
4	QB3.0	0xA480	QB19.0	0xA4C0	QW38.0	0xA540	QW70.0	0xA580	QD108.0	0xA640	QD172.0	0xA680
5	QB4.0	0xA480	QB20.0	0xA4C0	QW40.0	0xA540	QW72.0	0xA580	QD112.0	0xA640	QD174.0	0xA680
6	QB5.0	0xA480	QB21.0	0xA4C0	QW42.0	0xA540	QW74.0	0xA580	QD116.0	0xA640	QD180.0	0xA680
7	QB6.0	0xA480	QB22.0	0xA4C0	QW44.0	0xA540	QW76.0	0xA580	QD120.0	0xA640	QD184.0	0xA680
8	QB7.0	0xA480	QB23.0	0xA4C0	QW46.0	0xA540	QW78.0	0xA580	QD124.0	0xA640	QD188.0	0xA680
9	QB8.0	0xA480	QB24.0	0xA4C0	QW48.0	0xA540	QW80.0	0xA580	QD128.0	0xA640	QD192.0	0xA680
10	QB9.0	0xA480	QB25.0	0xA4C0	QW50.0	0xA540	QW82.0	0xA580	QD132.0	0xA640	QD196.0	0xA680
11	QB10.0	0xA480	QB26.0	0xA4C0	QW52.0	0xA540	QW84.0	0xA580	QD136.0	0xA640	QD200.0	0xA680
12	QB11.0	0xA480	QB27.0	0xA4C0	QW54.0	0xA540	QW86.0	0xA580	QD140.0	0xA640	QD204.0	0xA680
13	QB12.0	0xA480	QB28.0	0xA4C0	QW56.0	0xA540	QW88.0	0xA580	QD144.0	0xA640	QD208.0	0xA680
14	QB13.0	0xA480	QB29.0	0xA4C0	QW58.0	0xA540	QW90.0	0xA580	QD148.0	0xA640	QD212.0	0xA680
15	QB14.0	0xA480	QB31.0	0xA4C0	QW60.0	0xA540	QW92.0	0xA580	QD152.0	0xA640	QD216.0	0xA680
16	QB15.0	0xA480	QB31.0	0xA4C0	QW62.0	0xA540	QW94.0	0xA580	QD156.0	0xA640	QD220.0	0xA680

Table 7-72 Process Outputs – IEC 61131 Declaration Examples with Variables

8 Error Handling

8.1 Programming Environment Error Codes

Programming environment errors (which also include warnings) will be displayed in a popup window, provided that the programming tool is active. They will have the following effects:

- An error will stop the application program.
- A warning will only be signalled, but does not stop the application program.

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[*1)]
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

Error Code	Description	Comment
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	

Remark

1) EPOS Studio uses a watchdog reset also for resetting the Node. Therefore, this warning may also be triggered when EPOS Studio manipulates the EPOS P.

Table 8-73 Error Codes – Programming Environment

8.2 Motion Control Function Blocks Error Codes

Motion control function blocks can return internal error codes as well as error codes – such as e.g. communication aborted – of 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 8-74 Error Codes – Motion Control Function Blocks

9 Example Projects

9.1 «HelloWorld»

Project	HelloWorld	
Description	This example project provides a simple way to get used with the programming environment. Neither motion control functionality is used, nor must a motor be connected. The program may 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	Project file Main program Additional information	HelloWorld.VAR Counter.ST ReadMe.TXT

Table 9-75 «HelloWorld» in Brief

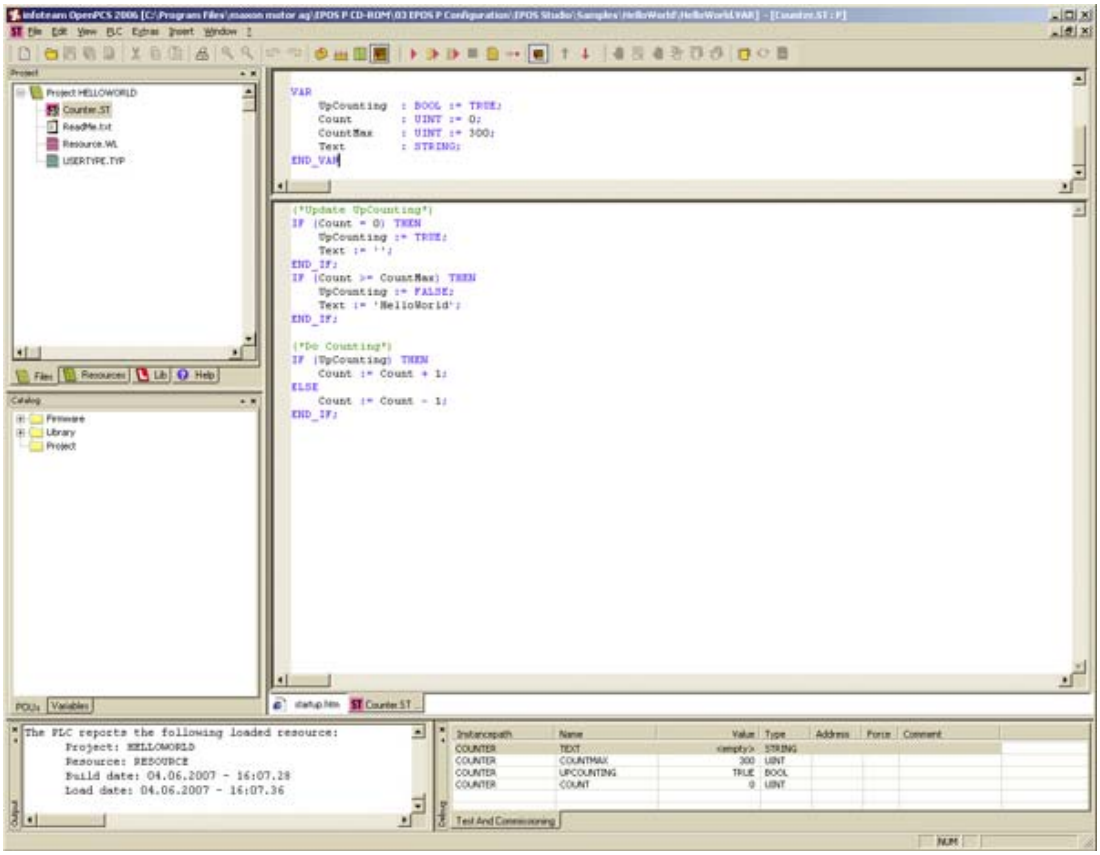


Table 9-76 «HelloWorld» – Project Screen

9.2 «SimpleMotionSequence»

Project	SimpleMotionSequence	
Description	The example consists of two state machines: • The first implements the application process. • The second implements error handling. The main state machine moves between two positions. For details ➔separate document «SimpleMotionSequence.pdf».	
Used Languages	SFC (Sequential Function Chart) FBD (Function Block Diagram)	
Task	Cyclic	
Files	Project file Main program Additional information	SimpleMotionSequence.VAR PROG_Main.SFC PROG_ErrorHandling.SFC

Table 9-77 «SimpleMotionSequence» in Brief

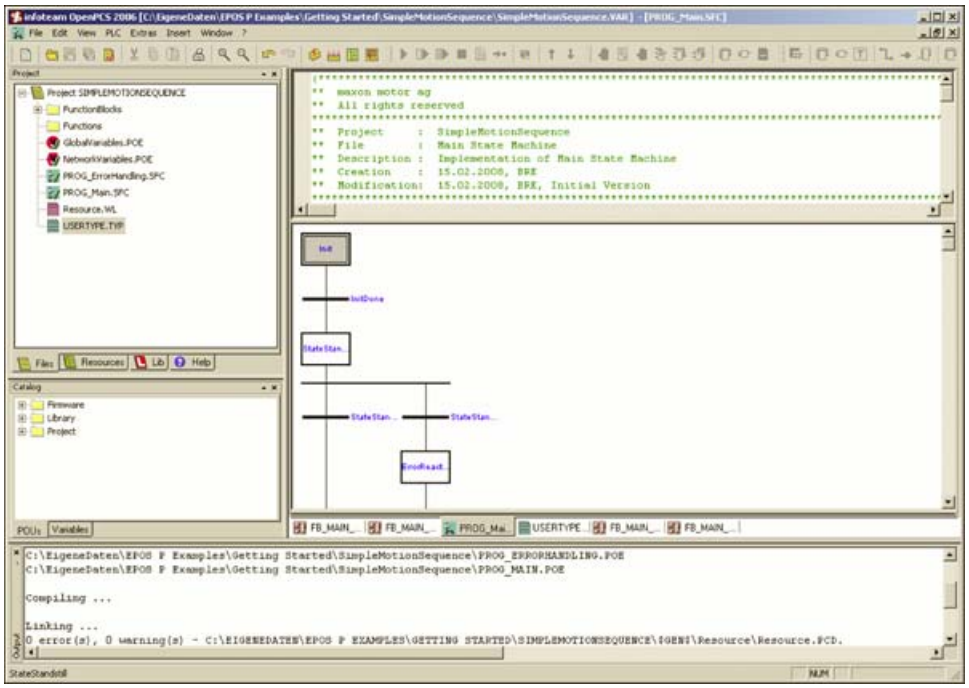


Table 9-78 «SimpleMotionSequence» – Project Screen

9.3 Best Practice Program Examples

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

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

Table 9-79 Best Practice Program Examples

9.4 Application Program Examples

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

Example	Description
«Cyclic Motion»	The example demonstrates typical motion sequences with one axis. It features homing, continuous motion and positioning. For details ➔separate document «CyclicMotionProject.pdf».
«I/O Mode»	The example demonstrates I/O triggered motions with one axis. For details ➔separate document «IO_ModeProject.pdf».
«Multi-Axis Motion»	The example demonstrates how to implement coordinated motions with two axes. For details ➔separate document «MultiaxisMotionProject.pdf».

Table 9-80 Application Program Examples

LIST OF FIGURES

LIST OF TABLES

INDEX

N

O

P

R

S

T

V

••page intentionally left blank••

maxon motor ag
Brünigstrasse 220
P.O.Box 263
CH-6072 Sachseln
Switzerland

Phone +41 (41) 666 15 00
Fax +41 (41) 666 15 50

www.maxonmotor.com