EPOS2 P

Programmable Positioning Controllers

Programming Reference

Document ID: rel3959

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

The EPOS2 P is 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**!

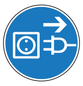
Examples:

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 Sources for additional Information

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

#	Reference
[1]	CiA 301 Communication Profile for Industrial Systems www.can-cia.org
[2]	CiA 302 Framework for CANopen Managers and Programmable CANopen Devices www.can-cia.org (section accessible for CiA members only)
[3]	CiA 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: EPOS2 Firmware Specification EPOS Positioning Controller DVD or www.maxonmotor.com
[7]	maxon motor: EPOS2 P Firmware Specification EPOS Positioning Controller DVD or www.maxonmotor.com
[8]	maxon motor: EPOS2 P Supervisory Control Reference.chm EPOS Positioning Controller DVD or www.maxonmotor.com

Table 1-2 Sources for additional Information

1.6 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.

Brand Name	Trademark Owner
CANopen® CiA®	© CiA CAN in Automation e.V, DE-Nuremberg
Windows®	© Microsoft Corporation, USA-Redmond, WA

Table 1-3 Brand Names and Trademark Owners

1.7 Copyright

The present document – including all parts thereof – is protected by copyright. Any use (including reproduction, translation, microfilming and other means of electronic data processing) beyond the narrow restrictions of the copyright law without the prior approval of maxon motor ag, is not permitted and subject to persecution under the applicable law.

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

Phone +41 41 666 15 00
Fax +41 41 666 16 50

www.maxonmotor.com

2 Introduction

2.1 Important Notice: Prerequisites for Permission to commence Installation

The EPOS2 P is 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 EPOS2 P Programmable Positioning Controllers. 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 EPOS2 P Programmable Positioning Controllers also on the Internet: ➔www.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:

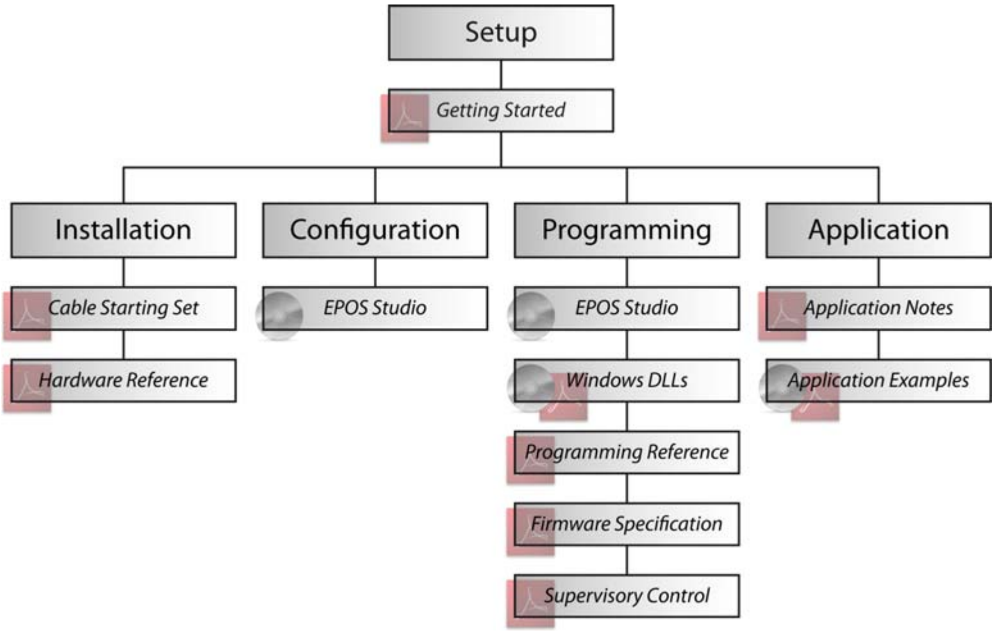


Figure 2-1 Documentation Structure

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.*

Electrostatic Sensitive Device (ESD)

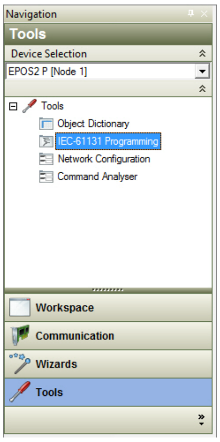
- *Make sure to wear working cloth in compliance with ESD countermeasures.*
- *Handle device with extra care.*

3 Programming

3.1 Programming Tool «OpenPCS»

3.1.1 Startup

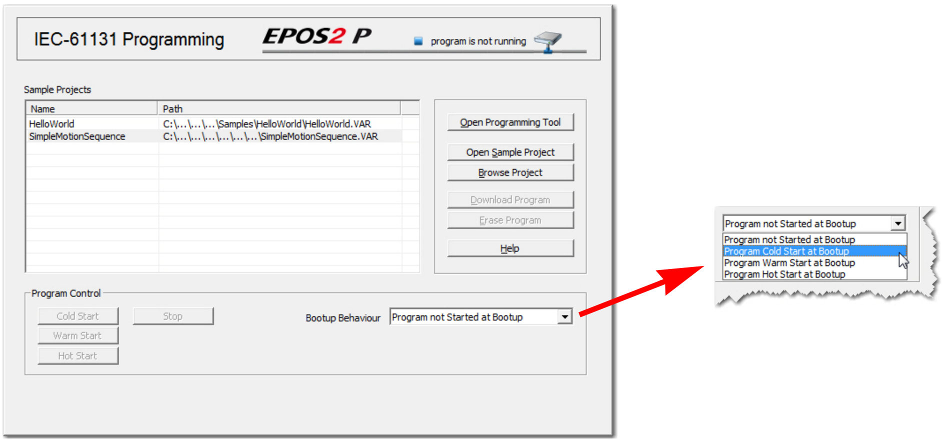
1) Open «EPOS Studio».
2) Load a project (*.pjm), containing a programmable controller permitting you to open the programming tool.
3) Click page ¤Tools¤ in page navigator.



Device Selection: EPOS2 P [Node 1]

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).



Name	Path
HelloWorld	C:\...\...\...\Samples\HelloWorld\HelloWorld.VAR
SimpleMotionSequence	C:\...\...\...\...\...\...\SimpleMotionSequence.VAR

Bootup Behaviour: Program not Started at Bootup

Figure 3-3 IEC 61131 Programming Windows

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	Download Program	Select and download an IEC 61131 project (*.var)
	Erase Program	Clear the IEC 61131 program on EPOS2 P
	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 behaviour after power up

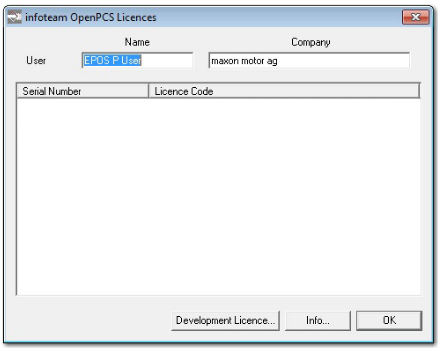
Table 3-4 IEC 61131 Programming Window – Commands and their Effect

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¤ and click menu item ¤Licence¤.
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).



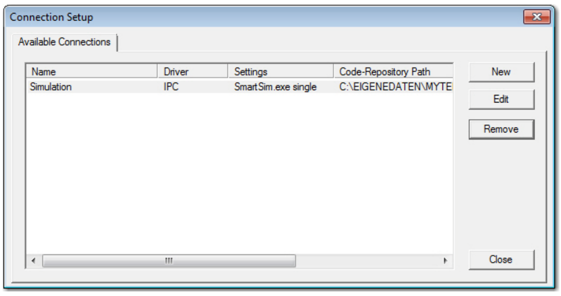
User Name: EPOS P User
Company: maxon motor ag

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

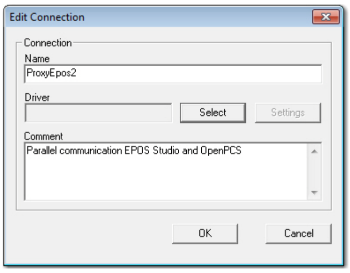
1) Open menu ¤PLC¤, then click menu item ¤Connections¤.



Name	Driver	Settings	Code-Repository Path
Simulation	IPC	SmartSim.exe single	C:\EIGENEDATEN\MYTE

Figure 3-5 Connection Setup

2) Look for one of the entries “ProxyEpos2_USB”, “ProxyEpos2_RS232”, “ProxyEpos2_CAN”.
   a) **If available**, click ¤Edit¤ and continue with step 5.
   b) **If not available**, click ¤New¤ and continue with next step.
3) Enter “ProxyEpos2” as name and add comments – later on, this driver will enable parallel communication of «EPOS Studio» and programming tool «OpenPCS». Then click ¤OK¤.



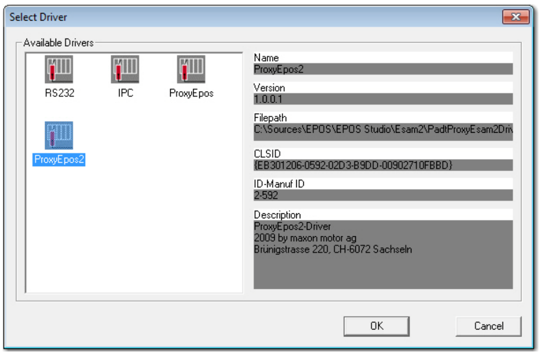
Name: ProxyEpos2

Driver:

Comment: Parallel communication EPOS Studio and OpenPCS

Figure 3-6 Edit Connection

4) Select ¤ProxyEpos2¤ to select driver. Then click ¤OK¤.



Name: ProxyEpos2

Version: 1.0.0.1

Filepath: C:\Sources\EPOS\EPOS Studio\Esam2\PadtProxyEsam2Driv

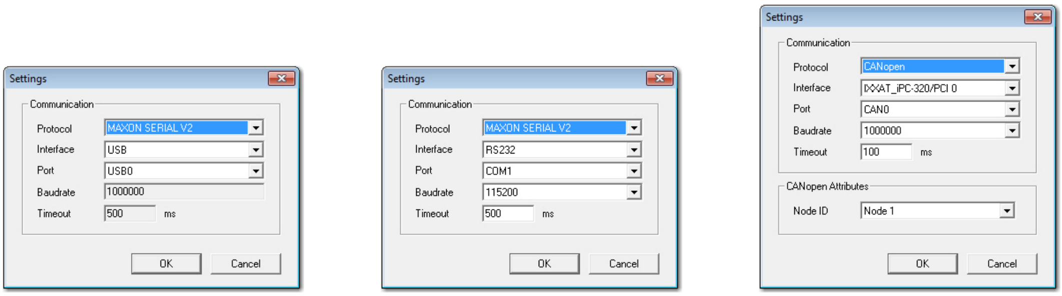
CLSID: {EB301206-0592-02D3-B9DD-00902710FBBD}

ID-Manuf ID: 2-592

Description: ProxyEpos2-Driver
2009 by maxon motor ag
Brünigstrasse 220, CH-6072 Sachseln

Figure 3-7 Select Driver

5) Select communication settings (for details ➔Table 3-5). Then click ¤OK¤.



Protocol: MAXON SERIAL V2
Interface: USB
Port: USB0
Baudrate: 1000000
Timeout: 500 ms

Protocol: MAXON SERIAL V2
Interface: RS232
Port: COM1
Baudrate: 115200
Timeout: 500 ms

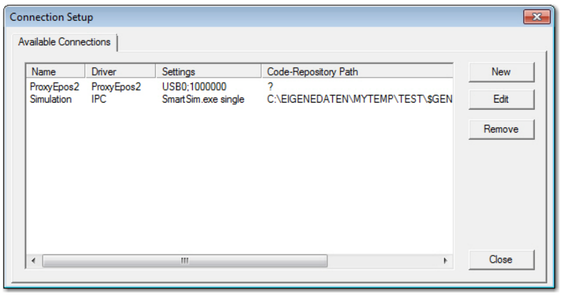
Protocol: CANopen
Interface: IXXAT_iPC-320/PCI 0
Port: CAN0
Baudrate: 1000000
Timeout: 100 ms
Node ID: Node 1

Figure 3-8 Connection Settings (USB, RS232, CANopen)

Area	Button / Command	Effect
Communi-cation	Protocol	Communication Protocol Stack to be used. **Range:** MAXON SERIAL V2, CANopen
	Interface	Communication Interface to be used. **Range:** USB, RS232, IXXAT, National Instruments, Vector
	Port	Communication Port to be used. **Range:** USBx, COMx, CANx
	Baudrate	Communication Baudrate to be used.
	Timeout	Communication Timeout. **Default:** 500 ms
CANopen Attributes	Node ID	Node Address for CANopen Communication. **Range:** Node 1…127

Table 3-5 Connection Settings – Commands

6) The connection entry has been added to the list and is available for selection. Click ¤Close¤ to close the window.



Name	Driver	Settings	Code-Repository Path
ProxyEpos2	ProxyEpos2	USB0;1000000	?
Simulation	IPC	SmartSim.exe single	C:\EIGENEDATEN\MYTEMP\TEST\$GEN

Figure 3-9 Connection Entry “ProxyEpos2”

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”.

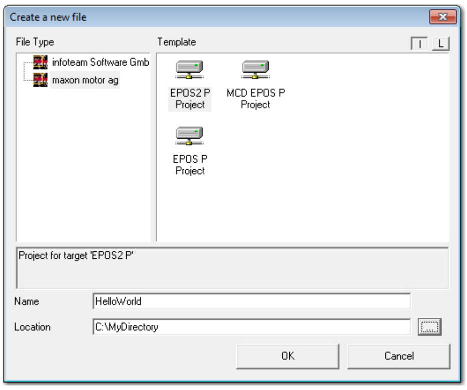
For an example using motion control functionality ➔chapter “9.2 «SimpleMotionSequence»” on page 9-144.

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 ¤Project¤. Select menu item ¤New¤.
2) Select file type ¤maxon motor ag¤ and template ¤EPOS2 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 EPOS2 P” and a network connection named “ProxyEpos2”.



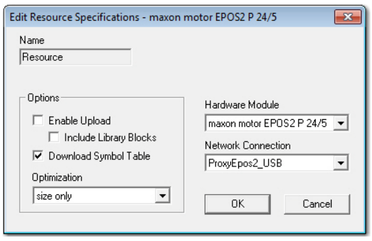
Project for target 'EPOS2 P'

Name: HelloWorld

Location: C:\MyDirectory

Figure 3-10 Create New Project

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



Name: Resource

☐ Enable Upload
☐ Include Library Blocks
☑ Download Symbol Table

Optimization: size only

Hardware Module: maxon motor EPOS2 P 24/5

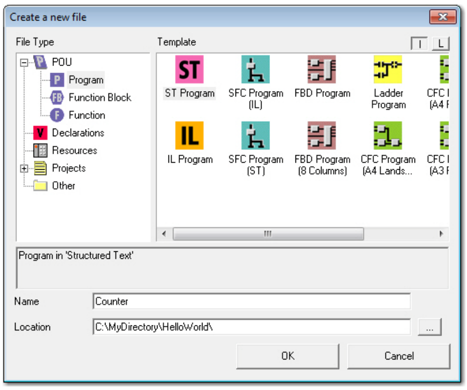
Network Connection: ProxyEpos2_USB

Figure 3-11 Edit Resource Specifications

3.5 Program Code

3.5.1 Writing Program Code

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



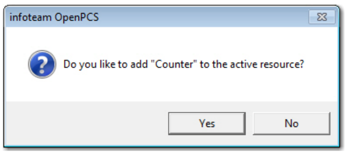
Program in 'Structured Text'

Name: Counter

Location: C:\MyDirectory\HelloWorld\

Figure 3-12 Create Program File

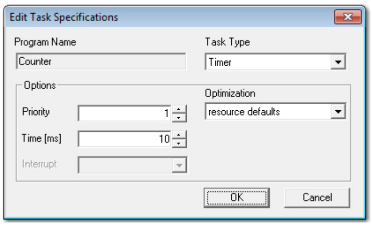
2) Select file type ¤Program¤ from directory “POU” (Program Organization Unit):
   a) Choose preferred programming language for your program – in following example “Structured Text”.
   b) Enter name “Counter” and click ¤OK¤.
3) You will be asked whether or not you wish to add program item “Counter”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



Do you like to add "Counter" to the active resource?

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.



Program Name: Counter

Task Type: Timer

Priority: 1

Time [ms]: 10

Optimization: resource defaults

Figure 3-14 Task Specifications

5) Now, you are ready to start programming:
   a) Open program item ¤Counter.ST¤.

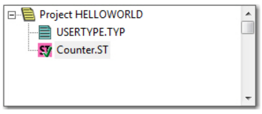
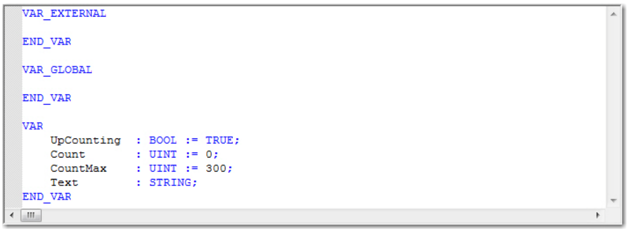


Figure 3-15 Project HelloWorld

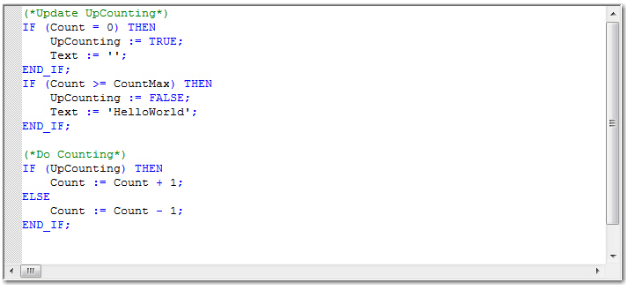
   b) Enter variable declaration.



```
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.



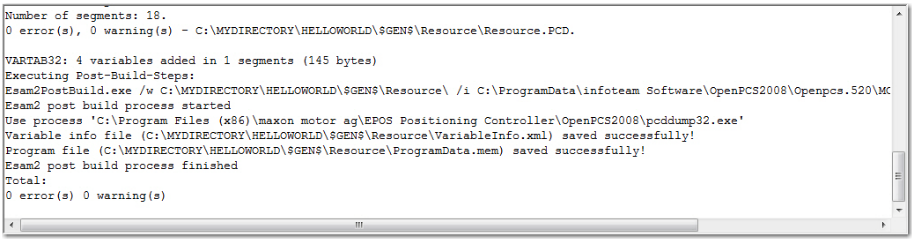
```
(*Update UpCounting*)
IF (Count = 0) THEN
    UpCounting := TRUE;
    Text := '';
END_IF;
IF (Count >= CountMax) THEN
    UpCounting := FALSE;
    Text := 'HelloWorld';
END_IF;

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

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

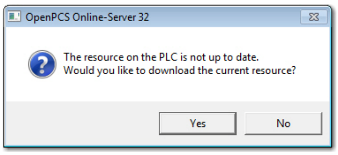
1) After code implementation, the program must be compiled:
   Click menu ¤PLC¤, then select menu item ¤Build Active Resource¤. The following logging output will be displayed.



```
Number of segments: 18.
0 error(s), 0 warning(s) - C:\MYDIRECTORY\HELLOWORLD\$GEN$\Resource\Resource.PCD.

VARTAB32: 4 variables added in 1 segments (145 bytes)
Executing Post-Build-Steps:
Esam2PostBuild.exe /w C:\MYDIRECTORY\HELLOWORLD\$GEN$\Resource\ /i C:\ProgramData\infoteam Software\OpenPCS2008\Openpcs.520\M
Esam2 post build process started
Use process 'C:\Program Files (x86)\maxon motor ag\EPOS Positioning Controller\OpenPCS2008\pcddump32.exe'
Variable info file (C:\MYDIRECTORY\HELLOWORLD\$GEN$\Resource\VariableInfo.xml) saved successfully!
Program file (C:\MYDIRECTORY\HELLOWORLD\$GEN$\Resource\ProgramData.mem) saved successfully!
Esam2 post build process finished
Total:
0 error(s) 0 warning(s)
```

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 EPOS2 P.



The resource on the PLC is not up to date.
Would you like to download the current resource?

Figure 3-19 Download new Code

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

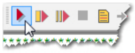
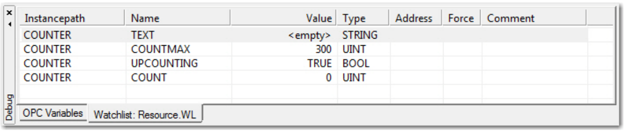


Figure 3-20 Cold Start

3.5.3 Debugging Program Code

1) Add a watch variable to the “Debug” window:
    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 the “Debug” window.

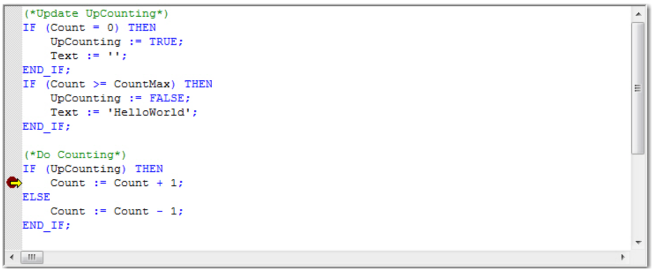


Instancepath	Name	Value	Type	Address	Force	Comment
COUNTER	TEXT	<empty>	STRING			
COUNTER	COUNTMAX	300	UINT			
COUNTER	UPCOUNTING	TRUE	BOOL			
COUNTER	COUNT	0	UINT			

Figure 3-21 “Debug” 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.



```
(*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;
```

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¤.

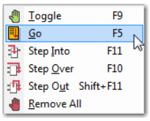


Figure 3-23 Continue Program Execution

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.

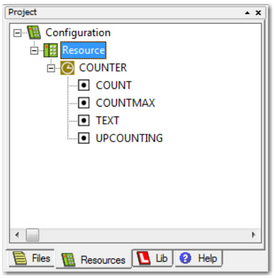
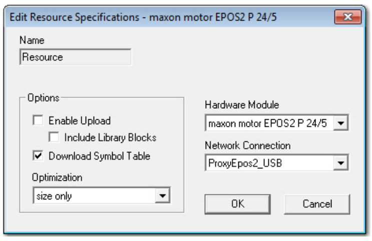


Figure 4-24 Resource Pane

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 (for details ➔Table 4-6):


Edit Resource Specifications - maxon motor EPOS2 P 24/5

Name: Resource

Options
☐ Enable Upload
☐ Include Library Blocks
☑ Download Symbol Table

Optimization: size only

Hardware Module: maxon motor EPOS2 P 24/5

Network Connection: ProxyEpos2_USB

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 following modules will be available: • “maxon motor EPOS2 P 24/5” • “maxon motor EPOS P 24/5” • “maxon motor MCD EPOS P 60 W” If you wish to use Windows SmartSIM simulation, select “SmartSIM”.
Network Connection	Select the communication connection to your resource. To communicate with maxon controllers, choose as follows: EPOS2 P 24/5: “ProxyEpos2” EPOS P 24/5: “ProxyEpos” MCD EPOS P 60 W: “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 4-6 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…

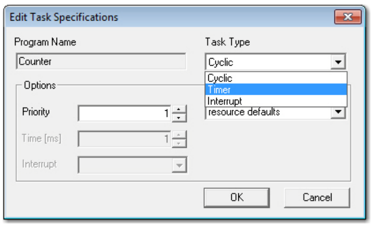
- name,
- information on the execution of the task, and
- POU of type PROGRAM that will be executed in this task.

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.

Task Type

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



Edit Task Specifications

Program Name: Counter

Task Type: Cyclic

Priority: 1

Time [ms]: 1

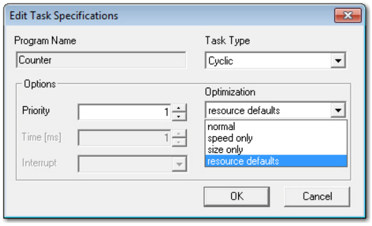
Figure 4-26 Task Type Window

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 4-7 Task Type Window – Control Elements

Optimization

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



Edit Task Specifications

Program Name: Counter

Task Type: Cyclic

Priority: 1

Time [ms]: 1

Optimization: resource defaults

Figure 4-27 Edit Task Specification – Optimization

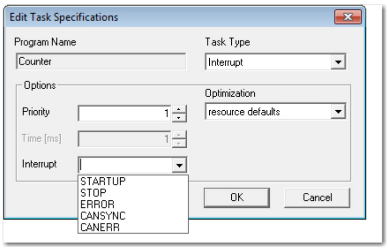
Control Element	Description
resource defaults	Uses the optimization attributes of the resource.
size only	Compiler option to optimize the generated code in respect to its size.
speed only	Compiler option to optimize the generated code in respect to speed.
normal	Mix between size only and speed only.

Table 4-8 Edit Task Specification – Optimization Control Elements

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

Interrupt

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



Edit Task Specifications

Program Name: Counter
Task Type: Interrupt
Options
Priority: 1
Time [ms]: 1
Interrupt:
Optimization: resource defaults

Figure 4-28 Edit Task Specification – Interrupt

Interrupt	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 4-9 Edit Task Specification – Interrupt Control Elements

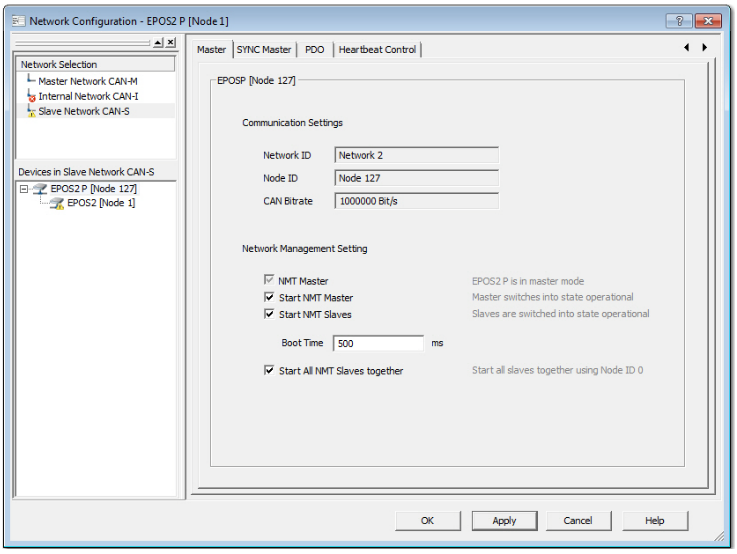
Remark

- *Interrupt Tasks “STARTUP”, “STOP” and “ERROR” need typically more than one cycle to finish!*
- *Interrupt Task “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*
- *Interrupt Task “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

This chapter explains the configuration for both, Internal Network (CAN-I) and Slave Network (CAN-S). For the configuration of a Master Network (CAN-M) ➔separate document «EPOS2 P Supervisory Control Reference».

4.3.1 Overview



Communication Settings

Network ID: Network 2
Node ID: Node 127
CAN Bitrate: 1000000 Bit/s

Network Management Setting

☑ NMT Master — EPOS2 P is in master mode
☑ Start NMT Master — Master switches into state operational
☑ Start NMT Slaves — Slaves are switched into state operational
Boot Time: 500 ms
☑ Start All NMT Slaves together — Start all slaves together using Node ID 0

Figure 4-29 Network Configuration Overview

Control Element	Description
Network Selection	Display of all available networks.
Device Selection in Network CAN-S	Display of all available devices within the selected network.
Tabs	Display of a particular configuration view to define parameters and settings.

Table 4-10 Network Configuration Overview – Display Elements

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

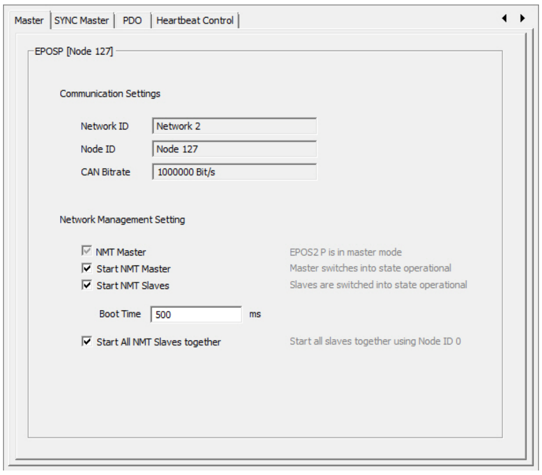
Table 4-11 Network Configuration Overview – Status & Icons

4.3.2 Master Configuration

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

4.3.2.1 Tab “Master”

Allows definition of behavior of the master device.



EPOSP [Node 127]

Communication Settings

Network ID: Network 2
Node ID: Node 127
CAN Bitrate: 1000000 Bit/s

Network Management Setting

☑ NMT Master — EPOS2 P is in master mode
☑ Start NMT Master — Master switches into state operational
☑ Start NMT Slaves — Slaves are switched into state operational

Boot Time: 500 ms

☑ Start All NMT Slaves together — Start all slaves together using Node ID 0

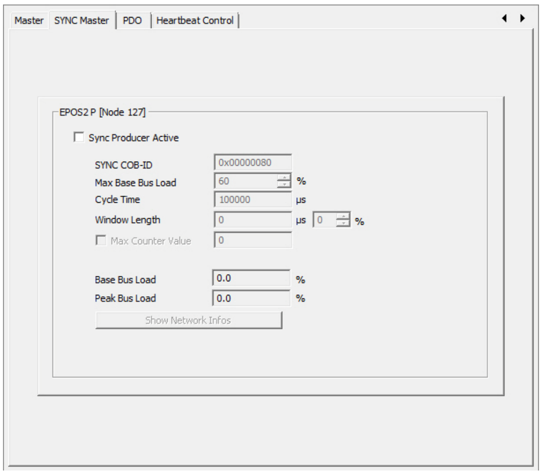
Figure 4-30 Tab “Master”

Area	Control Element	Description
Communication Settings	Network ID	Communication Network ID of the corresponding network.
	Node ID	Communication Node ID as a member of the corresponding network.
	CAN Bitrate	Communication Bitrate of the corresponding network.
Network Management Setting	NMT Master	EPOS2 P is in master mode and is able to communicate with slaves. **Default:** checked
	Start NMT Master	After bootup, the master is switching into NMT state operational. **Default:** checked
	Start NMT Slaves	After bootup, the master is switching the slaves into NMT state operational. **Default:** checked
	Boot Time	Time to wait before addressing slaves after reset. **Default:** 500 ms
	NMT Slaves together	All slaves are starting at the same time using a broadcast service. **Default:** checked

Table 4-12 Tab “Master” – Control Elements

4.3.2.2 Tab “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.



☐ Sync Producer Active

SYNC COB-ID 0x00000080
Max Base Bus Load 60 %
Cycle Time 100000 µs
Window Length 0 µs 0 %
☐ Max Counter Value 0

Base Bus Load 0.0 %
Peak Bus Load 0.0 %

Figure 4-31 Tab “SYNC Master”

Area	Control Element	Description
Sync Producer Active	Check box	Enable/disable the SYNC Master. **Default:** active
	SYNC COB-ID	COB-ID of the SYNC CAN Frame. **Default:** 0x00000080
	Max Base Bus Load	Recommended Maximum Base Bus Load. **Default:** 60%
	Cycle Time	Cycle Time of the SYNC CAN Frame. **Default:** 100’000 us
	Window Length	Window for sending and receiving synchronous PDOs. **Default:** 50%
	Max Counter Value	Enable or disable sending a SYNC CAN Frame including data byte containing a counter value. **Default:** disabled
	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 Tab “SYNC Master” – Options and Defaults/Calculations

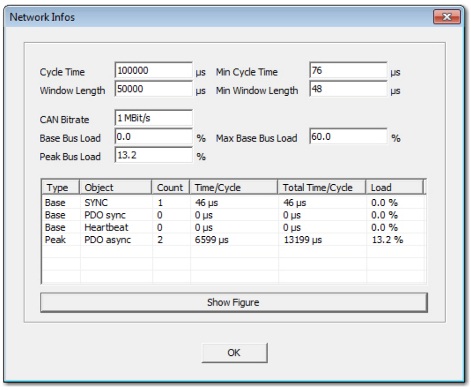
Best Practice: How to reduce Bus Load

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 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 Tab “SYNC Master” – Best Practice

For more details click ¤Show Network Infos¤:



Cycle Time 100000 µs Min Cycle Time 76 µs
Window Length 50000 µs Min Window Length 48 µs

CAN Bitrate 1 MBit/s
Base Bus Load 0.0 % Max Base Bus Load 60.0 %
Peak Bus Load 13.2 %

Type	Object	Count	Time/Cycle	Total Time/Cycle	Load
Base	SYNC	1	46 µs	46 µs	0.0 %
Base	PDO sync	0	0 µs	0 µs	0.0 %
Base	Heartbeat	0	0 µs	0 µs	0.0 %
Peak	PDO async	2	6599 µs	13199 µs	13.2 %

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. Consult the detailed load table for details on types of CAN frames that are included in the calculation .
Max Base Bus Load	Recommended Maximum Base Bus Load.
Peak Bus Load	Calculated bus load containing all CAN frames that are transmitted. Consult the detailed load table for details on types of CAN frames that are included in the calculation . **Remark**: Asynchronous PDOs are a potential risk for a bus overload. Use “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:

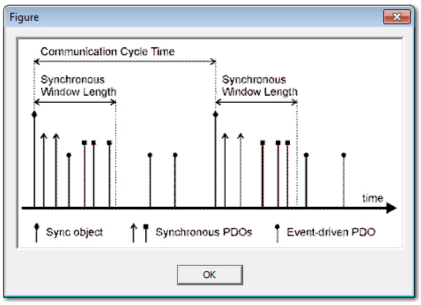
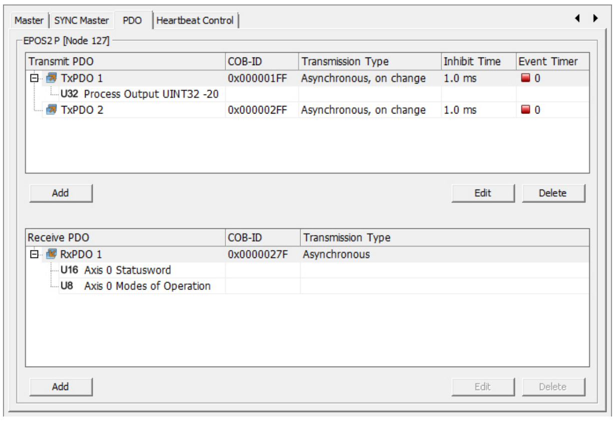


Figure 4-33 Cycle Time

4.3.2.3 Tab “PDO”

Used to edit and change the PDO configuration of the Master Network.

Configuration of network variables automatically adds PDOs and PDO Mappings
Make sure not to destroy the PDO configuration of a network variable!



Transmit PDO	COB-ID	Transmission Type	Inhibit Time	Event Timer
TxPDO 1	0x000001FF	Asynchronous, on change	1.0 ms	🟥 0
U32 Process Output UINT32 -20				
TxPDO 2	0x000002FF	Asynchronous, on change	1.0 ms	🟥 0

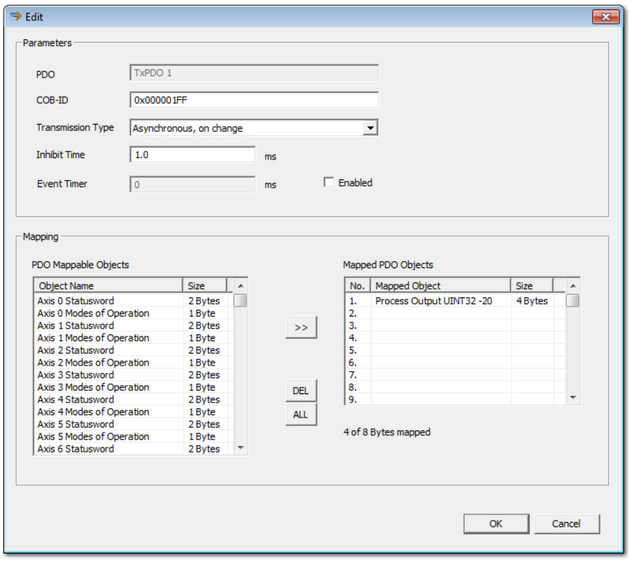
Receive PDO	COB-ID	Transmission Type
RxPDO 1	0x0000027F	Asynchronous
U16 Axis 0 Statusword		
U8 Axis 0 Modes of Operation		

Figure 4-34 Tab “PDO”

Area	Control Element	Description
Table Columns	Transmit PDO Receive PDO	PDOs and mapped object of the PDO
	COB-ID	11-Bit Identifier used by the PDO
	Transmission Type	defines the transmission/reception character of a PDO
	Inhibit Time	minimal transmission interval for asynchronous PDOs **Note!** An inhibit time of “0” (zero) represents a potential risk for bus overload!
	Event Timer	elapsed timer to trigger the asynchronous PDO transmission
Buttons	Add	to add a new Transmit/Receive PDO to the list **Note!** if inactive, no more PDOs can be added
	Edit	to change settings of an existing PDO
	Delete	to delete an existing PDO from the list

Table 4-17 Tab “PDO” – Functions

The dialog “Edit” displays the configuration options for Transmit and Receive PDOs.



PDO: TxPDO 1
COB-ID: 0x000001FF
Transmission Type: Asynchronous, on change
Inhibit Time: 1.0 ms
Event Timer: 0 ms ☐ Enabled

PDO Mappable Objects

Object Name	Size
Axis 0 Statusword	2 Bytes
Axis 0 Modes of Operation	1 Byte
Axis 1 Statusword	2 Bytes
Axis 1 Modes of Operation	1 Byte
Axis 2 Statusword	2 Bytes
Axis 2 Modes of Operation	1 Byte
Axis 3 Statusword	2 Bytes
Axis 3 Modes of Operation	1 Byte
Axis 4 Statusword	2 Bytes
Axis 4 Modes of Operation	1 Byte
Axis 5 Statusword	2 Bytes
Axis 5 Modes of Operation	1 Byte
Axis 6 Statusword	2 Bytes

Mapped PDO Objects

No.	Mapped Object	Size
1.	Process Output UINT32 -20	4 Bytes
2.		
3.		
4.		
5.		
6.		
7.		
8.		
9.		

4 of 8 Bytes mapped

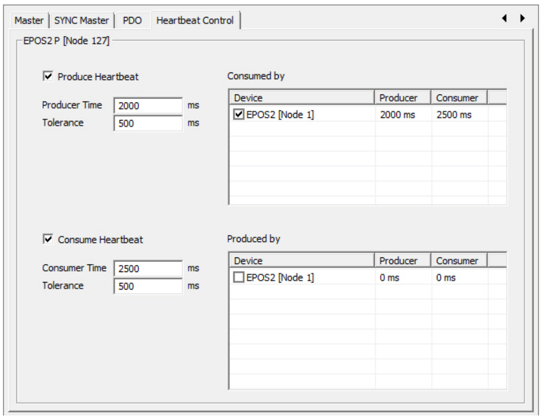
Figure 4-35 Tab “PDO” – Edit Dialog

Area	Control Element	Description
Parameters	PDO	name of PDO being configured
	COB-ID	11-Bit Identifier used by the PDO
	Transmission Type	defines the transmission/reception character of a PDO **Asynchronous:** PDO transmission is triggered by value change or event timer **Asynchronous RTR only:** PDO can be requested by a remote transfer request **Synchronous:** PDO transmission is triggered by the Sync Master
	Inhibit Time	minimal transmission interval for asynchronous PDOs **Note!** An inhibit time of “0” (zero) represents a potential risk for bus overload!
	Event Timer	elapsed timer to trigger the asynchronous PDO transmission
Mapping	PDO Mappable Objects	list of all objects that can be mapped to a PDO
	Mapped PDO Objects	list of all objects that are mapped to the PDO
Buttons	>>	to add an object to the PDO mapping
	DEL	to delete an object from the PDO mapping
	ALL	to delete all objects from the PDO mapping

Table 4-18 Tab “PDO” – Edit Dialog Functions

4.3.2.4 Tab “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.



EPOS2 P [Node 127]

☑ Produce Heartbeat

Producer Time 2000 ms
Tolerance 500 ms

Consumed by

Device	Producer	Consumer
☑ EPOS2 [Node 1]	2000 ms	2500 ms

☑ Consume Heartbeat

Consumer Time 2500 ms
Tolerance 500 ms

Produced by

Device	Producer	Consumer
☐ EPOS2 [Node 1]	0 ms	0 ms

Figure 4-36 Tab “Heartbeat Control”

Option	Default	Description
Produce Heartbeat	Producer Heartbeat	Enable or disable the heartbeat producer. **Default:** disabled
	Producer Time	Transmission rate of the heartbeat CAN frame. **Default:** 2000 ms
	Tolerance	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. **Default:** 500 ms
	Consumed by	**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 **Default:** disabled
Consume Heartbeat	Consumer Heartbeat	Enable or disable the heartbeat consumer. **Default:** disabled
	Consumer Time	Expected transmission rate of the heartbeat CAN frame. **Default:** 2000 ms
	Tolerance	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. **Default:** 500 ms
	Produced by	**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 **Default:** disabled

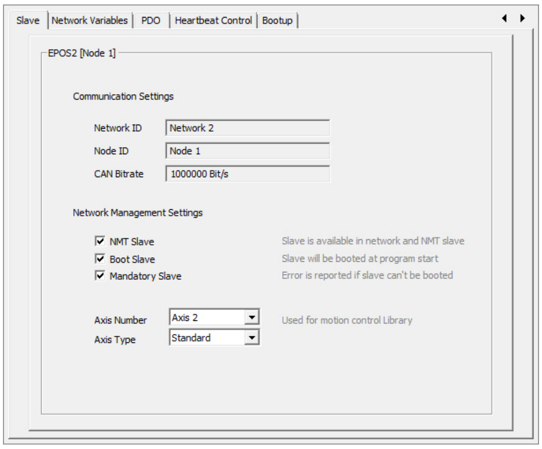
Table 4-19 Tab “Heartbeat Control” – Control Elements

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 Tab “Slave”

Allows to define the behavior of the slave device.



EPOS2 [Node 1]

Communication Settings

Network ID: Network 2
Node ID: Node 1
CAN Bitrate: 1000000 Bit/s

Network Management Settings

☑ NMT Slave — Slave is available in network and NMT slave
☑ Boot Slave — Slave will be booted at program start
☑ Mandatory Slave — Error is reported if slave can't be booted

Axis Number: Axis 2 — Used for motion control Library
Axis Type: Standard

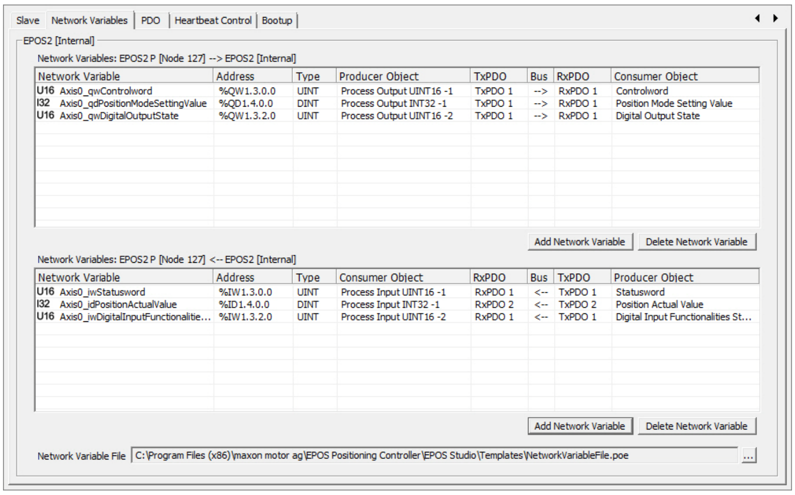
Figure 4-37 Tab “Slave”

Area	Control Element	Description
Communi-cation Settings	Network ID	Communication Network ID of the corresponding network.
	Node ID	Communication Node ID as a member of the corresponding network.
	CAN Bitrate	Communication Bitrate of the corresponding network.
Network Management Setting	NMT Slave	The slave is available in CAN network as a NMT slave. **Default:** checked
	Boot Slave	The slave will be booted at the program start. **Default:** checked
	Mandatory Slave	Error is reported if slave can’t be booted. **Default:** checked
	Axis Number	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. **Default:** Axis X
	Axis Type	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. **Default:** standard

Table 4-20 Tab “Slave” – Control Elements

4.3.3.2 Tab “Network Variables”

Allows to setup network variables for the IEC 61131 program.



Network Variables: EPOS2 P [Node 127] --> EPOS2 [Internal]

Network Variable	Address	Type	Producer Object	TxPDO	Bus	RxPDO	Consumer Object
U16 Axis0_qwControlword	%QW1.3.0.0	UINT	Process Output UINT16 -1	TxPDO 1	-->	RxPDO 1	Controlword
I32 Axis0_qdPositionModeSettingValue	%QD1.4.0.0	DINT	Process Output INT32 -1	TxPDO 1	-->	RxPDO 1	Position Mode Setting Value
U16 Axis0_qwDigitalOutputState	%QW1.3.2.0	UINT	Process Output UINT16 -2	TxPDO 1	-->	RxPDO 1	Digital Output State

Network Variables: EPOS2 P [Node 127] <-- EPOS2 [Internal]

Network Variable	Address	Type	Consumer Object	RxPDO	Bus	TxPDO	Producer Object
U16 Axis0_iwStatusword	%IW1.3.0.0	UINT	Process Input UINT16 -1	RxPDO 1	<--	TxPDO 1	Statusword
I32 Axis0_idPositionActualValue	%ID1.4.0.0	DINT	Process Input INT32 -1	RxPDO 2	<--	TxPDO 2	Position Actual Value
U16 Axis0_iwDigitalInputFunctionalitie...	%IW1.3.2.0	UINT	Process Input UINT16 -2	RxPDO 1	<--	TxPDO 1	Digital Input Functionalities St...

Network Variable File: C:\Program Files (x86)\maxon motor ag\EPOS Positioning Controller\EPOS Studio\Templates\NetworkVariableFile.poe

Figure 4-38 Tab “Network Variables”

Network Variables: EPOS2 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 4-21 Network Variables: EPOS2 P [Node 1] to EPOS [Internal]

Network Variables: EPOS2 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.

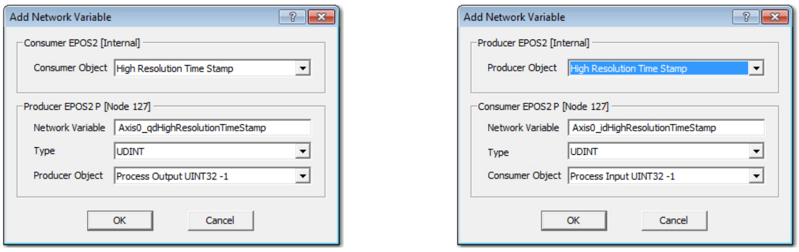
Table 4-22 Network Variables: EPOS2 P [Node 1] from EPOS [Internal]

Control Element	Description
Add Network Variable	Adds a network variable
Delete Network Variable	Deletes the selected network variable.

Table 4-23 Tab “Network Variables” – Control Elements

Add Network Variable

Click ¤Add Network Variable¤ button.



Consumer EPOS2 [Internal]
Consumer Object: High Resolution Time Stamp
Producer EPOS2 P [Node 127]
Network Variable: Axis0_qdHighResolutionTimeStamp
Type: UDINT
Producer Object: Process Output UINT32 -1

Producer EPOS2 [Internal]
Producer Object: High Resolution Time Stamp
Consumer EPOS2 P [Node 127]
Network Variable: Axis0_idHighResolutionTimeStamp
Type: UDINT
Consumer Object: Process Input UINT32 -1

Figure 4-39 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.
	Type	IEC 61131 type process variable – if “BOOL”, bit number is required.
	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.
	Type	IEC 61131 type process variable – if “BOOL”, bit number is required.
	Consumer Object	Process object of master.

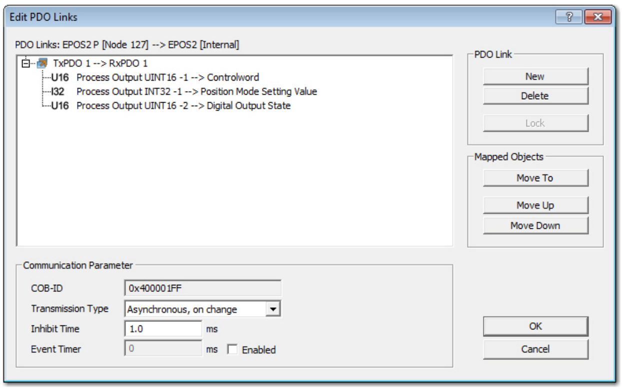
Table 4-24 Add Network Variable – Parameters

Edit PDO Links

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

.

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.



PDO Links: EPOS2 P [Node 127] --> EPOS2 [Internal]

- TxPDO 1 --> RxPDO 1
  - U16 Process Output UINT16 -1 --> Controlword
  - I32 Process Output INT32 -1 --> Position Mode Setting Value
  - U16 Process Output UINT16 -2 --> Digital Output State

COB-ID: 0x400001FF
Transmission Type: Asynchronous, on change
Inhibit Time: 1.0 ms
Event Timer: 0 ms ☐ Enabled

Figure 4-40 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 RTR only**: Do not use for network variables. **Asynchronous on event**: The PDO transmission is triggered by the IEC-61131 FB “CAN_SetTxPdoEvent”. **Asynchronous on change**: The PDO transmission is triggered by a value change.
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 4-25 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 4-26 Edit PDO Links – PDO Link

Mapped Objects

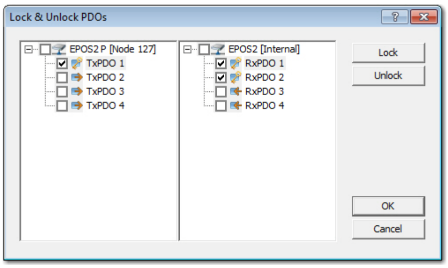
Control Element	Description
Move To	Move the selected objects to another PDO link.
Move Up	Move the selected objects up in the list of mapped objects.
Move Down	Move the selected object down in the list of mapped objects.

Table 4-27 Edit PDO Links – Mapped Objects

Lock/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 Lock & Unlock PDOs



☐ EPOS2 P [Node 127]
- ☑ TxPDO 1
- ☐ TxPDO 2
- ☐ TxPDO 3
- ☐ TxPDO 4

☐ EPOS2 [Internal]
- ☑ RxPDO 1
- ☑ RxPDO 2
- ☐ RxPDO 3
- ☐ RxPDO 4

Figure 4-41 Lock/unlock PDOs

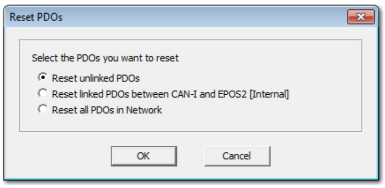
Icon	Description
🔑 Locked PDO	Cannot be used by any other network variables.
→ Unlocked transmit PDO	Can be used by new network variables.
← Unlocked receive PDO	Can be used by new network variables.

Table 4-28 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 Reset PDOs



Select the PDOs you want to reset

● Reset unlinked PDOs
○ Reset linked PDOs between CAN-I and EPOS2 [Internal]
○ Reset all PDOs in Network

Figure 4-42 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 EPOS2 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 4-29 Reset PDOs – Options

Show Network Variable File

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

Right click

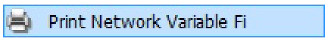
Save Network Variable File

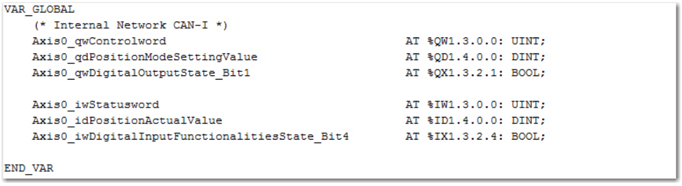
The declarations of the network variables for the IEC 61131 program are saved to a file (*.poe). This file can be included in a IEC 61131 program.

Right click

Print Network Variable File

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

Right click 



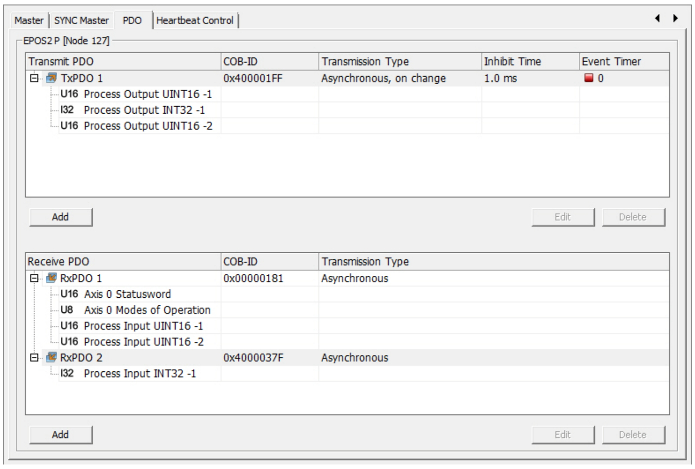
```
VAR_GLOBAL
    (* Internal Network CAN-I *)
    Axis0_qwControlword                              AT %QW1.3.0.0: UINT;
    Axis0_qdPositionModeSettingValue                 AT %QD1.4.0.0: DINT;
    Axis0_qwDigitalOutputState_Bit1                  AT %QX1.3.2.1: BOOL;

    Axis0_iwStatusword                               AT %IW1.3.0.0: UINT;
    Axis0_idPositionActualValue                      AT %ID1.4.0.0: DINT;
    Axis0_iwDigitalInputFunctionalitiesState_Bit4    AT %IX1.3.2.4: BOOL;

END_VAR
```

Figure 4-43 Declaration of Network Variables

4.3.3.3 Tab “PDO”

Used to edit and change the PDO configuration of the Master Network.

Configuration of network variables automatically adds PDOs and PDO Mappings
Make sure not to destroy the PDO configuration of a network variable!



Transmit PDO	COB-ID	Transmission Type	Inhibit Time	Event Timer
TxPDO 1	0x400001FF	Asynchronous, on change	1.0 ms	🟥 0
U16 Process Output UINT16 -1				
I32 Process Output INT32 -1				
U16 Process Output UINT16 -2				

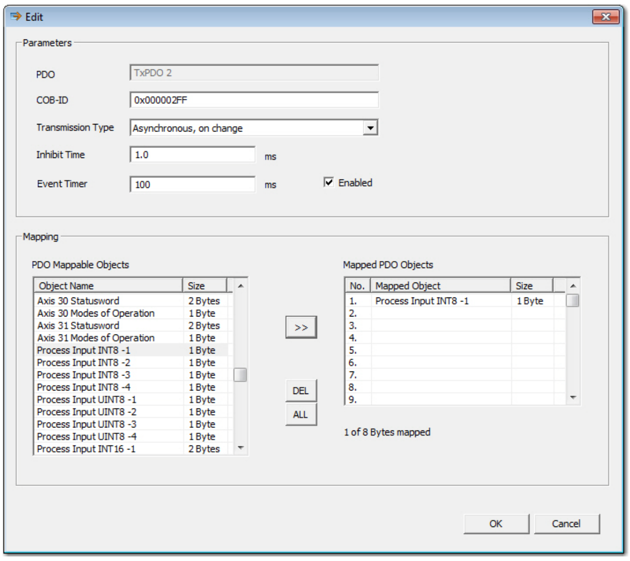
Receive PDO	COB-ID	Transmission Type
RxPDO 1	0x00000181	Asynchronous
U16 Axis 0 Statusword		
U8 Axis 0 Modes of Operation		
U16 Process Input UINT16 -1		
U16 Process Input UINT16 -2		
RxPDO 2	0x4000037F	Asynchronous
I32 Process Input INT32 -1		

Figure 4-44 Tab “PDO”

Area	Control Element	Description
Table Columns	Transmit PDO Receive PDO	PDOs and mapped object of the PDO
	COB-ID	11-Bit Identifier used by the PDO
	Transmission Type	defines the transmission/reception character of a PDO
	Inhibit Time	minimal transmission interval for asynchronous PDOs **Note!** An inhibit time of “0” (zero) represents a potential risk for bus overload!
	Event Timer	elapsed timer to trigger the asynchronous PDO transmission
Buttons	Add	to add a new Transmit/Receive PDO to the list **Note!** if inactive, no more PDOs can be added
	Edit	to change settings of an existing PDO
	Delete	to delete an existing PDO from the list

Table 4-30 Tab “PDO” – Functions

The dialog “Edit” displays the configuration options for Transmit and Receive PDOs.



PDO: TxPDO 2
COB-ID: 0x000002FF
Transmission Type: Asynchronous, on change
Inhibit Time: 1.0 ms
Event Timer: 100 ms ☑ Enabled

PDO Mappable Objects

Object Name	Size
Axis 30 Statusword	2 Bytes
Axis 30 Modes of Operation	1 Byte
Axis 31 Statusword	2 Bytes
Axis 31 Modes of Operation	1 Byte
Process Input INT8 -1	1 Byte
Process Input INT8 -2	1 Byte
Process Input INT8 -3	1 Byte
Process Input INT8 -4	1 Byte
Process Input UINT8 -1	1 Byte
Process Input UINT8 -2	1 Byte
Process Input UINT8 -3	1 Byte
Process Input UINT8 -4	1 Byte
Process Input INT16 -1	2 Bytes

Mapped PDO Objects

No.	Mapped Object	Size
1.	Process Input INT8 -1	1 Byte
2.		
3.		
4.		
5.		
6.		
7.		
8.		
9.		

1 of 8 Bytes mapped

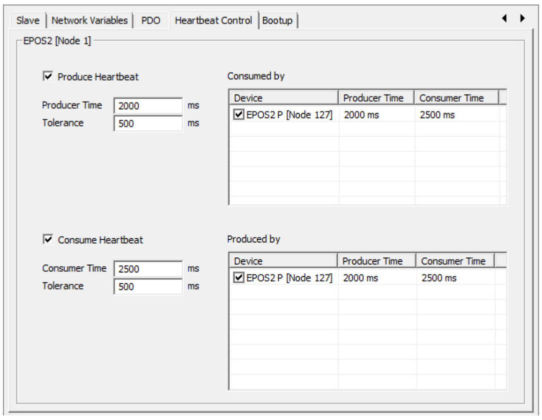
Figure 4-45 Tab “PDO” – Edit Dialog

Area	Control Element	Description
Parameters	PDO	name of PDO being configured
	COB-ID	11-Bit Identifier used by the PDO
	Transmission Type	defines the transmission/reception character of a PDO **Asynchronous:** PDO transmission is triggered by value change or event timer **Asynchronous RTR only:** PDO can be requested by a remote transfer request **Synchronous:** PDO transmission is triggered by the Sync Master
	Inhibit Time	minimal transmission interval for asynchronous PDOs **Note!** An inhibit time of “0” (zero) represents a potential risk for bus overload!
	Event Timer	elapsed timer to trigger the asynchronous PDO transmission
Mapping	PDO Mappable Objects	list of all objects that can be mapped to a PDO
	Mapped PDO Objects	list of all objects that are mapped to the PDO
Buttons	>>	to add an object to the PDO mapping
	DEL	to delete an object from the PDO mapping
	ALL	to delete all objects from the PDO mapping

Table 4-31 Tab “PDO” – Edit Dialog Functions

4.3.3.4 Tab “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.



☑ Produce Heartbeat

Producer Time 2000 ms
Tolerance 500 ms

Consumed by

Device	Producer Time	Consumer Time
☑ EPOS2 P [Node 127]	2000 ms	2500 ms

☑ Consume Heartbeat

Consumer Time 2500 ms
Tolerance 500 ms

Produced by

Device	Producer Time	Consumer Time
☑ EPOS2 P [Node 127]	2000 ms	2500 ms

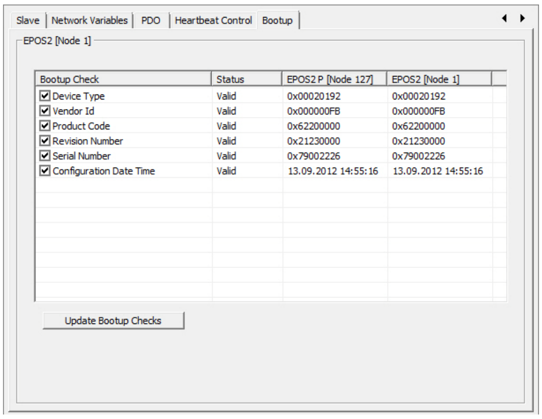
Figure 4-46 Tab “Heartbeat Control”

Area	Control Element	Description
Produce Heartbeat	Producer Heartbeat	Enable or disable the heartbeat producer. **Default:** disabled
	Producer Time	Transmission rate of the heartbeat CAN frame. **Default:** 2000 ms
	Tolerance	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. **Default:** 500 ms
	Consumed by	**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 **Default:** disabled
Consume Heartbeat	Consumer Heartbeat	Enable or disable the heartbeat consumer. **Default:** disabled
	Consumer Time	Expected transmission rate of the heartbeat CAN frame. **Default:** 2000 ms
	Tolerance	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. **Default:** 500 ms
	Produced by	**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 **Default:** disabled

Table 4-32 Tab “Heartbeat Control” – Control Elements

4.3.3.5 Tab “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.



EPOS2 [Node 1]

Bootup Check	Status	EPOS2 P [Node 127]	EPOS2 [Node 1]
☑ Device Type	Valid	0x00020192	0x00020192
☑ Vendor Id	Valid	0x000000FB	0x000000FB
☑ Product Code	Valid	0x62200000	0x62200000
☑ Revision Number	Valid	0x21230000	0x21230000
☑ Serial Number	Valid	0x79002226	0x79002226
☑ Configuration Date Time	Valid	13.09.2012 14:55:16	13.09.2012 14:55:16

Figure 4-47 Tab “Bootup”

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

Table 4-33 Tab “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.

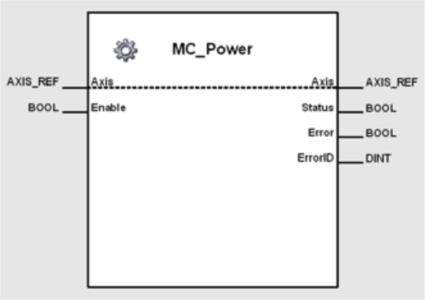
1) **Step 1: Create Project in EPOS Studio**
   a) Select menu item ¤New Project¤ in menu “File”.
   b) Select an EPOS2 P project template and click ¤Next¤.
   c) Enter project name, destination directory and click ¤Finish¤.
2) **Step 2: Scan the Network Topology**
   a) Change to tab ¤Communication¤ 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 ¤EPOS2 P¤ in device selection.
   c) Click item ¤Network Configuration¤ to open tool.
4) **Step 4: Minimal Master Configuration**
   a) Select master device ¤EPOS2 P¤ in device selection.
   b) Select tab “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 tab “SYNC Master” and disable Sync Producer.
   d) Select tab “Heartbeat Control” and disable Heartbeat Producer.
5) **Step 5: Minimal Slave Configuration**
   a) Select one of the slave devices in device selection.
   b) Select tab “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 tab “Heartbeat Control” and disable Heartbeat Producer.
   d) Select tab “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.

4.4 Communication

4.4.1 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

<table>
<tr><th colspan="2">Function Block</th><th colspan="2">Configuration</th></tr>
<tr><td>Parameter</td><td>AXIS_REF.AxisNo = Axis Number</td><td>Parameter</td><td>Axis Number</td></tr>
<tr><td colspan="2">Function Block Example


MC_Power
AXIS_REF
Axis
Axis
AXIS_REF
BOOL
Enable
Status
BOOL
Error
BOOL
ErrorID
DINT

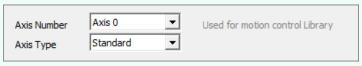
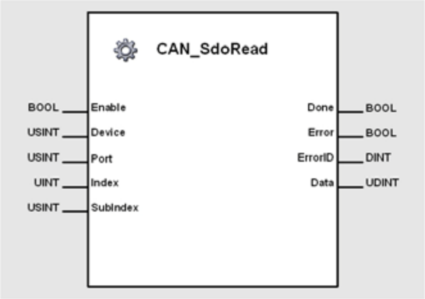
</td><td colspan="2">Tab “Slave”:

Axis Number: Axis 0
Axis Type: Standard
Used for motion control Library</td></tr>
</table>

Table 4-34 Motion Control Function Block: Configuration of Axis Number

CANopen CiA 301 Function Blocks

<table>
<tr><th colspan="2">Function Block</th><th colspan="2">Configuration</th></tr>
<tr><td>Parameter</td><td>Device: Node Id
Port: 0 internal port, 1 CAN port</td><td>Parameter</td><td>Node Id</td></tr>
<tr><td colspan="2">Function Block Example:


CAN_SdoRead
BOOL
Enable
Done
BOOL
USINT
Device
Error
BOOL
USINT
Port
ErrorID
DINT
UINT
Index
Data
UDINT
USINT
SubIndex

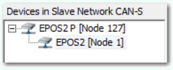
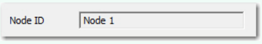
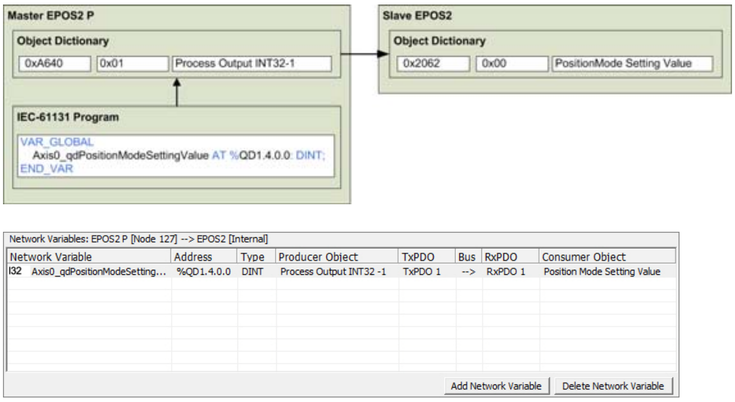
</td><td colspan="2">Device Selection:

Devices in Slave Network CAN-S
EPOS2 P [Node 127]
EPOS2 [Node 1]
Can be changed by DIP switch or Startup Wizard

Node ID: Node 1</td></tr>
</table>

Table 4-35 CANopen CiA 301 Function Block: Configuration of Node ID

4.4.2 Communication via Network Variables

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

1) **Step 1: Open tab “Network Variables”**
    a) Open tool “Network Configuration”.
    b) Select one of the slave devices in device selection and activate tab “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.



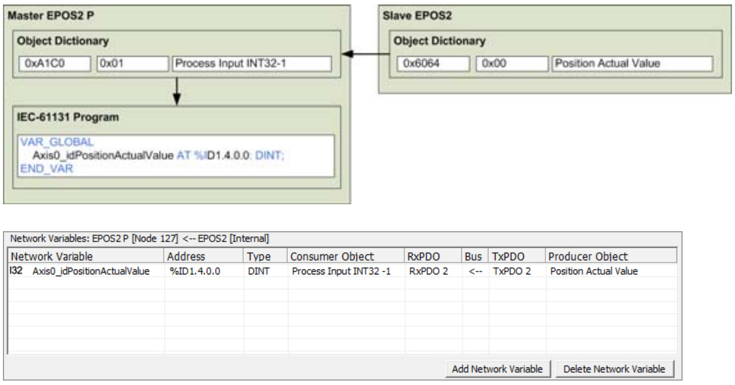
```
VAR_GLOBAL
   Axis0_qdPositionModeSettingValue AT %QD1.4.0.0: DINT;
END_VAR
```

Network Variables: EPOS2 P [Node 127] --> EPOS2 [Internal]

Network Variable	Address	Type	Producer Object	TxPDO	Bus	RxPDO	Consumer Object
I32 Axis0_qdPositionModeSetting...	%QD1.4.0.0	DINT	Process Output INT32 -1	TxPDO 1	-->	RxPDO 1	Position Mode Setting Value

Figure 4-48 Output Network Variables (from IEC 61131 Program to Slave)

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.



```
VAR_GLOBAL
   Axis0_idPositionActualValue AT %ID1.4.0.0: DINT;
END_VAR
```

Network Variables: EPOS2 P [Node 127] <-- EPOS2 [Internal]

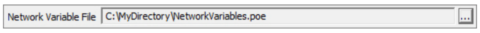
Network Variable	Address	Type	Consumer Object	RxPDO	Bus	TxPDO	Producer Object
I32 Axis0_idPositionActualValue	%ID1.4.0.0	DINT	Process Input INT32 -1	RxPDO 2	<--	TxPDO 2	Position Actual Value

Figure 4-49 Input Network Variables (from Slave to IEC 61131 Program)

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.



Network Variable File: C:\MyDirectory\NetworkVariables.poe

Figure 4-50 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.

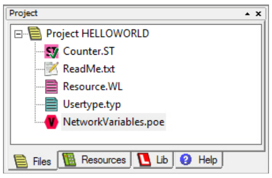


Figure 4-51 Project Browser in Programming Tool

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 Administrative

5.1.1.1 MC_Power

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

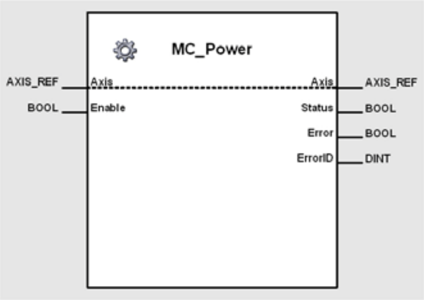


Figure 5-52 MC_Power

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

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)]	Status	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes →page 8-142	–

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 5-36 MC_Power

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);
```

5.1.1.2 MC_ReadStatus

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

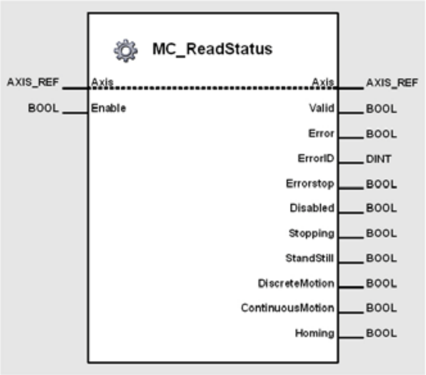


Figure 5-53 MC_ReadStatus

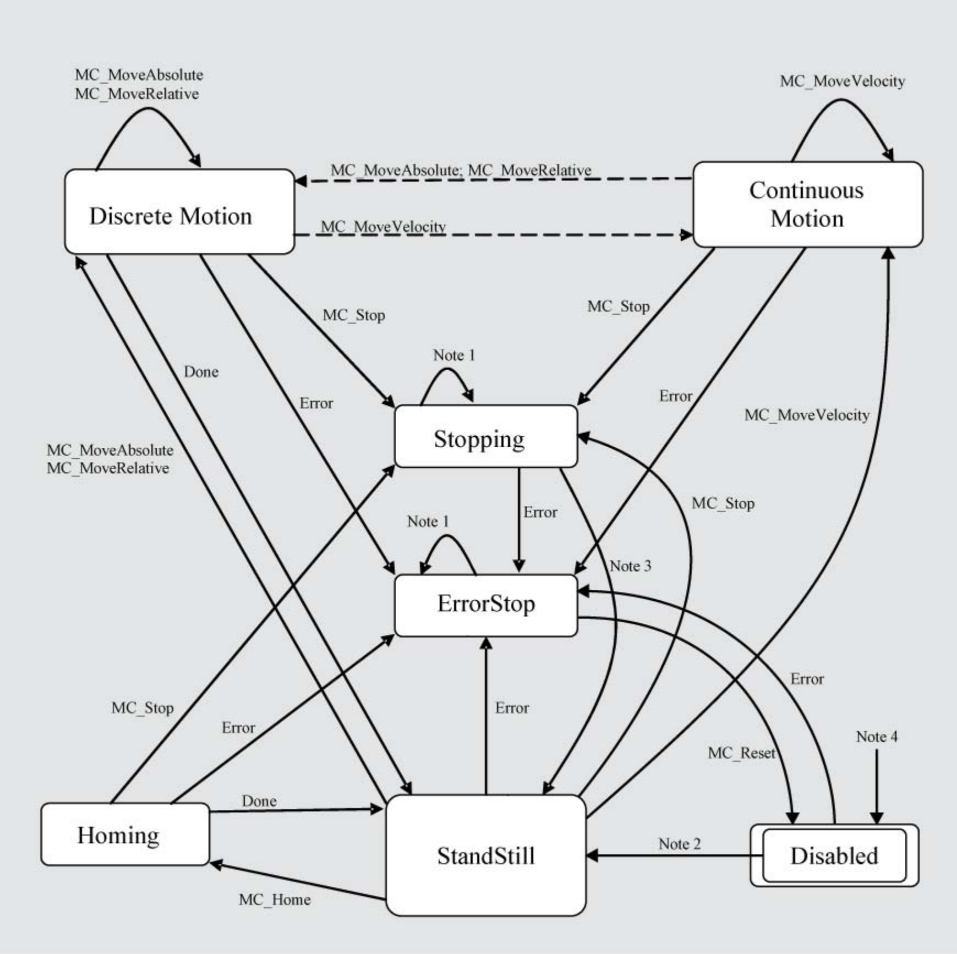
Variables

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)]	Valid	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-142	–
	Errorstop	BOOL	FALSE	TRUE, FALSE	–
	Disabled	BOOL	FALSE	TRUE, FALSE	–
	Stopping	BOOL	FALSE	TRUE, FALSE	–
	StandStill	BOOL	FALSE	TRUE, FALSE	–
	DiscreteMotion	BOOL	FALSE	TRUE, FALSE	–
	ContinuousMotion	BOOL	FALSE	TRUE, FALSE	–
	Homing	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 5-37 MC_ReadStatus

Details on possible states (➔Figure 5-54).



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 5-54 MC_ReadStatus – States

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;
```

5.1.1.3 MC_ReadAxisError

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

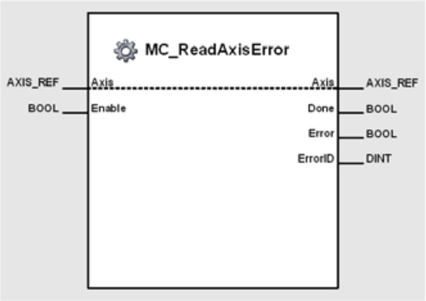


Figure 5-55 MC_ReadAxisError

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 1-7	–

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

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

Table 5-38 MC_ReadAxisError

5.1.1.4 MC_ReadParameter

Returns an axis parameter value.

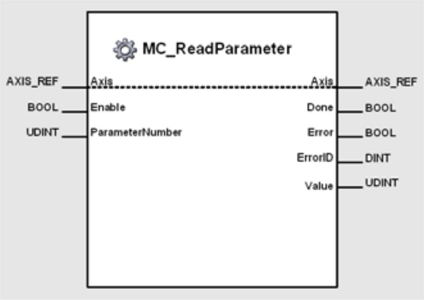


Figure 5-56 MC_ReadParameter

Important!
Execution of the instance might take longer than one PLC cycle (➔page 5-47).

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 subindex (hex) zz: Object length (hex)	–
Output[*O)]	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-142	–
	Value	UDINT	0	0…4’294’967’295	–

I) As long as *Enable* is TRUE (positive state), the value of a specified parameter is continuously being read.
ParameterNumber defines 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 «EPOS2 Firmware Specification»).
The multiplexer (for details ➔“Multplexer Example” on page 5-53) is composed of 2 bytes object index (Byte 3 and 2), 1 byte object subindex (Byte 1) and 1 byte object length (Byte 0).

O) *Value* allows retrieval of the value.

Table 5-39 MC_ReadParameter

Multplexer Example

ParameterNumber =	16#207C0102
Name =	Analog Input 1
Object Index =	16#207C
Object Subindex =	16#01
Object Length =	16#02

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);
```

5.1.1.5 MC_ReadLongParameter

Returns on 64-bit axia parameter value.

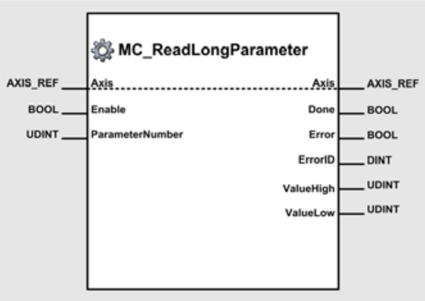


Figure 5-57 MC_ReadLongParameter

Important!
Execution of the instance might take longer than one PLC cycle (➔page 5-47).

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	16#xxxxyyzz multiplexer (hex) xxxx: Object index (hex) yy: Object subindex (hex) zz: Object length (hex)	–
Output*O)	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-142	–
	ValueHigh	UDINT	0	0…4’294’967’295	–
	ValueLow	UDINT	0	0…4’294’967’295	–

I) As long as *Enable* is TRUE (positive state), the value of a specified parameter is continuously being read.
ParameterNumber defines the parameter to be read. CANopen objects can be read using *ParameterNumber* as a multiplexer. Thus, allowing to read all EPOS objects from the object dictionary (➔separate document «EPOS2 Firmware Specification»).
The multiplexer (for details ➔“Multplexer Example” on page 5-55) is composed of 2 bytes object index (Byte 3 and 2), 1 byte object subindex (Byte 1) and 1 byte object length (Byte 0).

O) *ValueLow* and ValueHigh allows retrieval of a 64-Bit value.

Table 5-40 MC_ReadLongParameter

Multplexer Example

ParameterNumber =	16#20040008
Name =	Serial Number
Object Index =	16#2004
Object Subindex =	16#00
Object Length =	16#08

Call

```
--------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbReadP : MC_ReadLongParameter; (* fbReadP is instance of MC_ReadLongParameter *)
END_VAR
--------------------------------------------------------------------------------------
(* Function Block call for reading the CANopen object Serial Number*)
fbReadP(Axis := myAxis, Enable := TRUE, ParameterNumber := 16#20040008);
```

5.1.1.6 MC_ReadBoolParameter

Returns an axis parameter value.

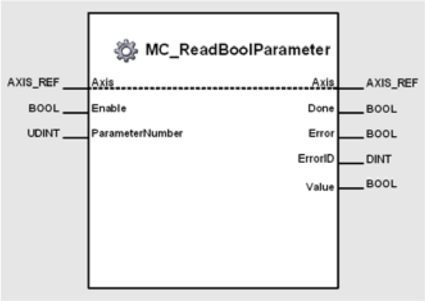


Figure 5-58 MC_ReadBoolParameter

Important!
Execution of the instance might take longer than one PLC cycle (➔page 5-47).

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 8-142	–
	Value	UDINT	0	0…4’294’967’295	–

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

O) *Value* allows retrieval of the value.

Table 5-41 MC_ReadBoolParameter

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 parameter “EnableLimitPos”*)
fbReadB(Axis := myAxis, Enable := TRUE, ParameterNumber := 4);
```

5.1.1.7 MC_WriteParameter

Modifies the value of an axis parameter.

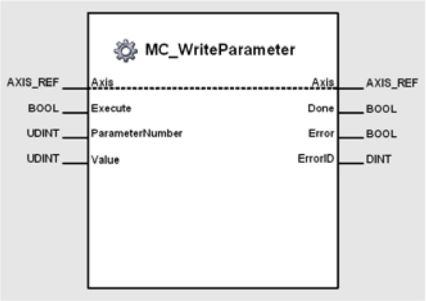


Figure 5-59 MC_WriteParameter

Important!
Execution of the instance might take longer than one PLC cycle (➔page 5-47).

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	–
	ParameterNumber	UDINT	0	**PLCopen parameter:** 2 SWLimitPos 3 SWLimitNeg 7 MaxPositionLag 8 MaxVelocitySystem 9 MaxVelocityAppl 11 CommandedVelocity 13 MaxAccelerationAppl 15 MaxDecelerationAppl 1000 SaveAllParameter **CANopen objects:** 16#xxxxyyzz multiplexer (hex) xxxx: Object index (hex) yy: Object subindex (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 8-142	–

I) A positive edge of *Execute* triggers a write operation of the specified parameter. *ParameterNumber* defines 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 «EPOS2 Firmware Specification»). The multiplexer (for details ➔“Multplexer Example” on page 5-58) is composed of 2 bytes object index (Byte 3 and 2), 1 byte object subindex (Byte 1) and 1 byte object length (Byte 0).

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

Table 5-42 MC_WriteParameter

Multplexer Example

ParameterNumber =	16#20780102
Name =	Analog Input 1
Object Index =	16#2078
Object Subindex =	16#01
Object Length =	16#02

Call

```
-----------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbWriteP : MC_WriteParameter; (* fbWriteP is instance of MC_WriteParameter *)
END_VAR
-----------------------------------------------------------------------------------
(* Function Block call for writing the digital outputs *)
fbWriteP(Axis := myAxis, Execute := TRUE, ParameterNumber := 16#20780102);
```

5.1.1.8 MC_WriteLongParameter

Modifies the value of a 64-bit axis parameters.

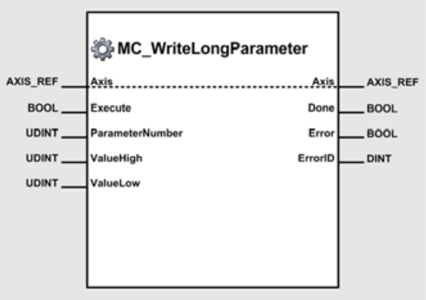


Figure 5-60 MC_WriteLongParameter

Important!
Execution of the instance might take longer than one PLC cycle (➔page 5-47).

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	–
	ParameterNumber	UDINT	0	16#xxxxyyzz multiplexer (hex) xxxx: Object index (hex) yy: Object subindex (hex) zz: Object length (hex)	–
	ValueHigh	UDINT	0	0…4’294’967’295	–
	ValueLow	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 8-142	–

I) A positive edge of *Execute* triggers a write operation of the specified parameter.
ParameterNumber defines the parameter to be written. CANopen objects can be read using *ParameterNumber* as a multiplexer. Thus, allowing to read all EPOS objects from the object dictionary (➔separate document «EPOS2 Firmware Specification»).
The multiplexer (for details ➔“Multplexer Example” on page 5-60) is composed of 2 bytes object index (Byte 3 and 2), 1 byte object subindex (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_WriteLongParameter

Multplexer Example

ParameterNumber =	16#20C10008
Name =	Interpolation Data Record
Object Index =	16#20C1
Object Subindex =	16#00
Object Length =	16#08

Call

```
------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbWriteP : MC_WriteLongParameter; (* fbWriteP is instance of MC_WriteLongParameter *)
END_VAR
------------------------------------------------------------------------------------
(* Function Block call for writing the interpolation data record *)
fbWriteP.ValueLow := 16#0000FFFF;
fbWriteP.ValueHigh := 16#00001000;
fbWriteP (Axis := myAxis, Execute := TRUE, ParameterNumber := 16#20C10008);
```

5.1.1.9 MC_ReadActualPosition

Returns the actual position of an axis.

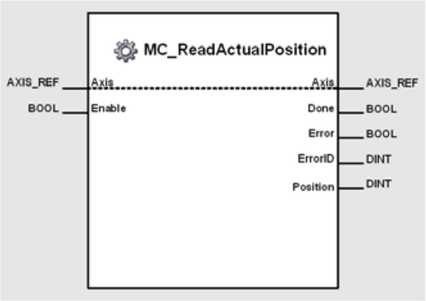


Figure 5-61 MC_ReadActualPosition

Important!
Execution of the instance might take longer than one PLC cycle (➔page 5-47).

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 8-142	–
	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 is continuously being read.
O) The actual position can be retrieved from *Position*.
Position is defined in quadcount (encoder increments) [qc].

Table 5-44 MC_ReadActualPosition

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);
```

5.1.1.10 MC_ReadActualVelocity

Returns the actual velocity of an axis.

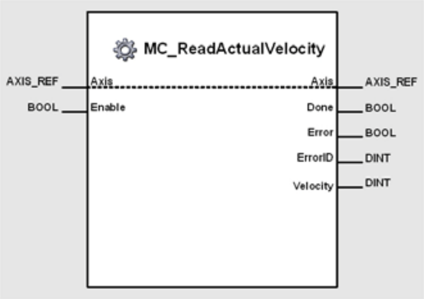


Figure 5-62 MC_ReadActualVelocity

Important!
Execution of the instance might take longer than one PLC cycle (➔page 5-47).

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 8-142	–
	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 is continuously being read.
O) The actual velocity can be retrieved from *Velocity*.

Table 5-45 MC_ReadActualVelocity

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);
```

5.1.1.11 MC_ReadActualCurrent

Returns the actual current of an axis.

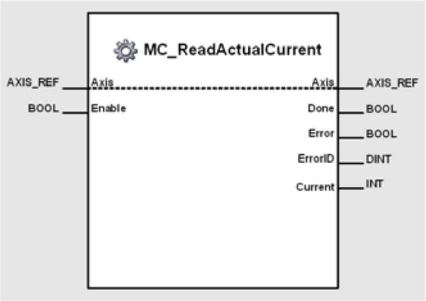


Figure 5-63 MC_ReadActualCurrent

Important!
Execution of the instance might take longer than one PLC cycle (➔page 5-47).

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 8-142	–
	Current	INT	0	- 32768 [min(INT)] … + 32767 [max(INT)]	mA

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

Table 5-46 MC_ReadActualCurrent

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);
```

5.1.1.12 MC_Reset

Resets all internal axis-related errors.

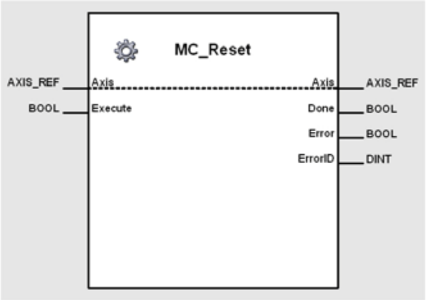


Figure 5-64 MC_Reset

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

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 8-142	–

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 5-48).
O) *Done* signals successful reset of axis status.

Table 5-47 MC_Reset

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);
```

5.1.1.13 MC_SetOperationMode

Sets the operation mode.

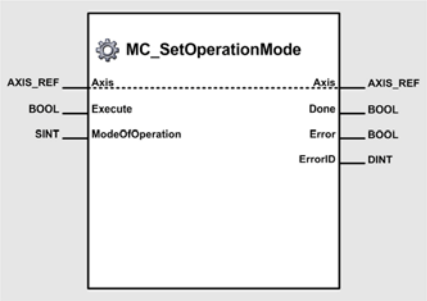


Figure 5-65 MC_SetOperationMode

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	–
	ModeOfOperation	SINT		Profile Position Mode = 1 Profile Velocity Mode = 3 Homing Mode = 6 Interpolated Position Mode = 7 Position Mode = -1 Velocity Mode = -2 Current Mode = -3 Master Encoder Mode = -5 Step/Direction Mode = -6	–
Output*O)	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes →page 8-142	–

I) A positive edge of *Execute* triggers a write operation of the operation mode object.
O) Successful write operation is signalled with a positive value (TRUE) at *Done*.

Table 5-48 MC_SetOperationMode

Call

```
-------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbSetOpMode : MC_SetOperationMode; (* fbSetOpMode is instance of MC_SetOperationMode
*)
END_VAR
-------------------------------------------------------------------------------------
(* Function Block call for writing the mode of operation to position mode *)
fbSetOpMode (Axis := myAxis, Execute := TRUE, ModeOfOperation := 16#FF);
```

5.1.2 Motion

5.1.2.1 MC_MoveAbsolute

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

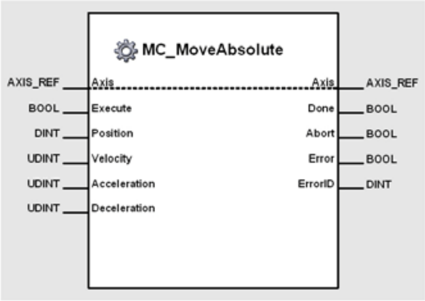


Figure 5-66 MC_MoveAbsolute

Important!
Execution of the instance might take longer than one PLC cycle (➔page 5-47).

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
	Velocity	UDINT	0	0…max. profile velocity	rpm
	Acceleration	UDINT	0	0…max. acceleration	rpm/s
	Deceleration	UDINT	0	0…max. deceleration	rpm/s
Output$^{*O)}$	Done	BOOL	FALSE	TRUE, FALSE	–
	Abort	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-142	–

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 5-49 MC_MoveAbsolute

Details on possible calling sequences (➔Figure 5-67).

- 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.

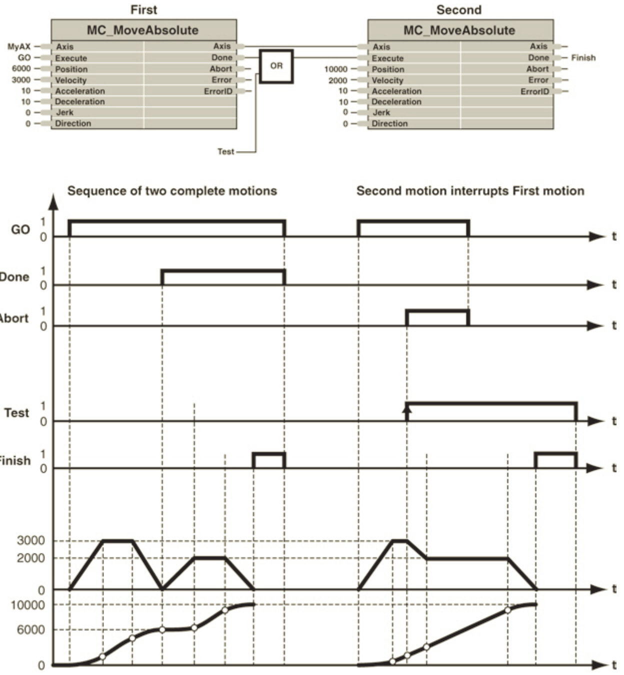


Figure 5-67 MC_MoveAbsolute – Sequence

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,Deceleration:=50);
```

5.1.2.2 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 last position setting value.

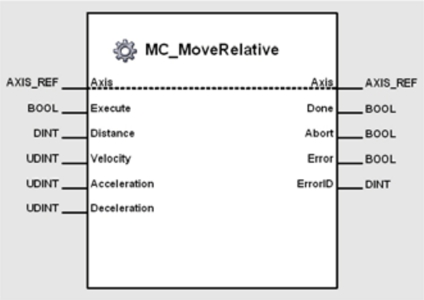


Figure 5-68 MC_MoveRelative

Important!
Execution of the instance might take longer than one PLC cycle (➔page 5-47).

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	–
	Distance	DINT	0	-2‘147‘483‘648…+2‘147‘483‘647	qc
	Velocity	UDINT	0	0…25‘000	rpm
	Acceleration	UDINT	0	0…4‘294‘967’295	rpm/s
	Deceleration	UDINT	0	0…4‘294‘967’295	rpm/s
Output[*O)]	Done	BOOL	FALSE	TRUE, FALSE	–
	Abort	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-142	–

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 last position setting value and commanded as a new 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 5-50 MC_MoveRelative

Details on possible calling sequences (➔Figure 5-69).

- 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.

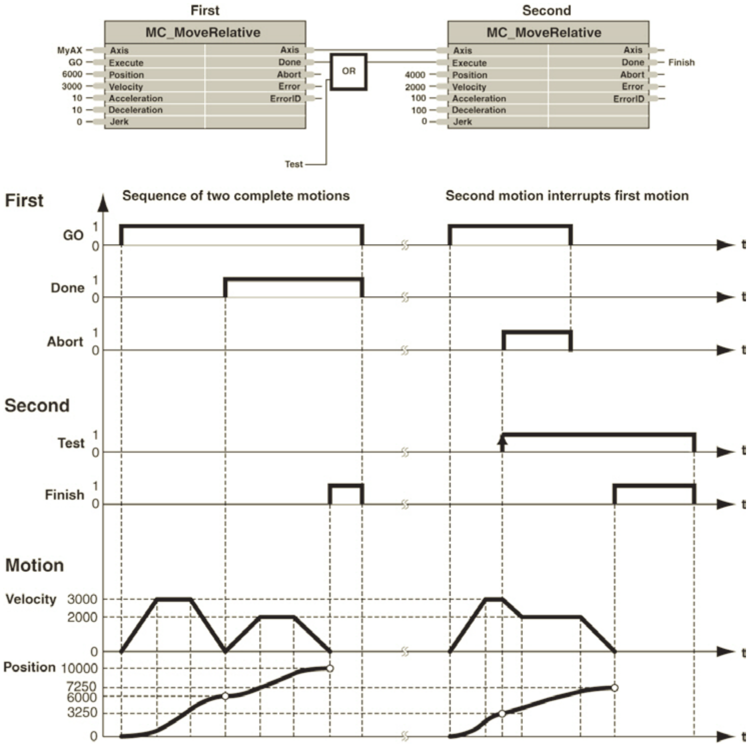


Figure 5-69 MC_MoveRelative – Sequence

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);
```

5.1.2.3 MC_MoveVelocity

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

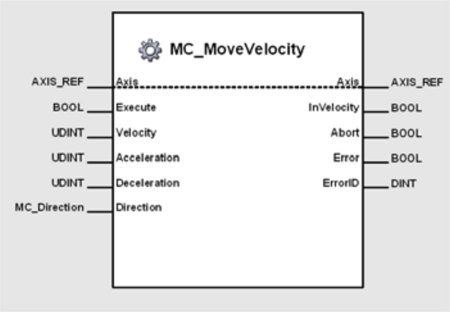


Figure 5-70 MC_MoveVelocity

Important!
Execution of the instance might take longer than one PLC cycle (→page 5-47).

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	–
	Velocity	UDINT	0	0…25‘000	rpm
	Acceleration	UDINT	0	0…4‘294‘967’295	rpm/s
	Deceleration	UDINT	0	0…4‘294‘967’295	rpm/s
	Direction	Enum MC_Direction	MCposi-tive	MCpositive MCnegative	–
Output [*O)]	InVelocity	BOOL	FALSE	TRUE, FALSE	–
	Abort	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes →page 8-142	–

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 5-51 MC_MoveVelocity

Details on possible calling sequences (➔Figure 5-71).

- 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.

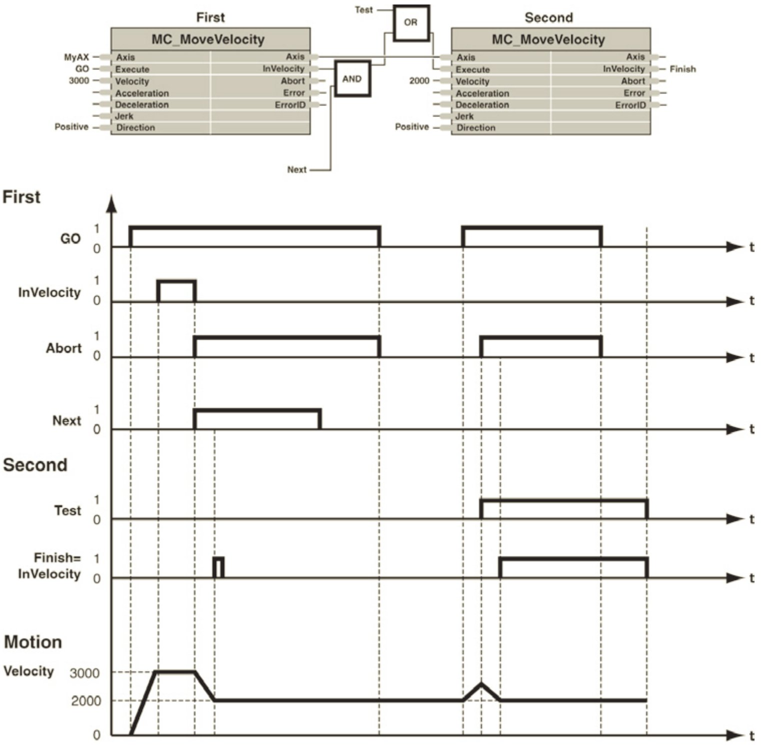


Figure 5-71 MC_MoveVelocity – Sequence

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);
```

5.1.2.4 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 «EPOS2 Firmware Specification»).

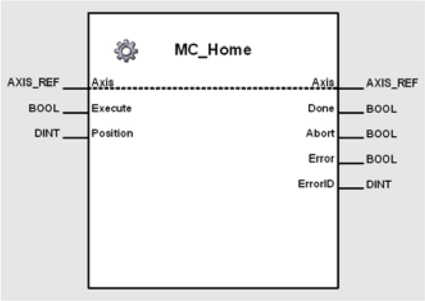


Figure 5-72 MC_Home

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$^{*O)}$	Done	BOOL	FALSE	TRUE, FALSE	–
	Abort	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-142	–

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 5-57), for detailed information ➔separate document «EPOS2 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 5-52 MC_Home

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);
```

5.1.2.5 MC_Stop

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

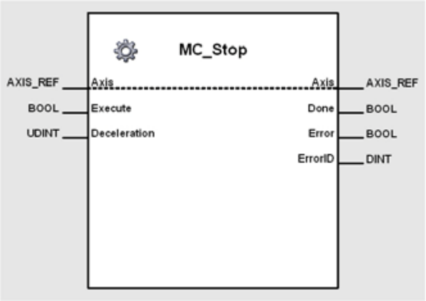


Figure 5-73 MC_Stop

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…max. acceleration	rpm/s
Output*O)	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-142	–

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 5-53 MC_Stop

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);
```

5.2 Maxon Utility Function Blocks

5.2.1 Homing

5.2.1.1 MU_GetHomingParameter

Returns the values of the EPOS homing objects.

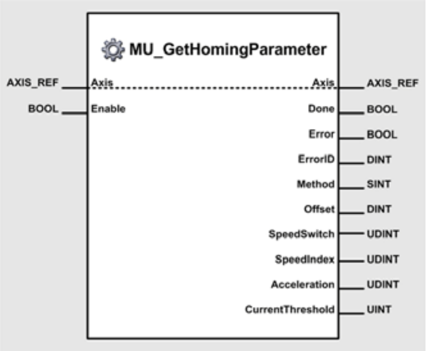


Figure 5-74 MU_GetHomingParameter

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 8-142	–
	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
	SpeedSwitch	UDINT	100	0…max. profile velocity	rpm
	SpeedIndex	UDINT	100	0…max. profile velocity	rpm
	Acceleration	UDINT	1000	0…max. acceleration	rpm/s
	CurrentThreshold	UINT	500	0 and up (depending on hardware)	mA

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

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

Table 5-54 MU_GetHomingParameter

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);
```

5.2.1.2 MU_SetHomingParameter

Modifies the values of the EPOS homing objects.

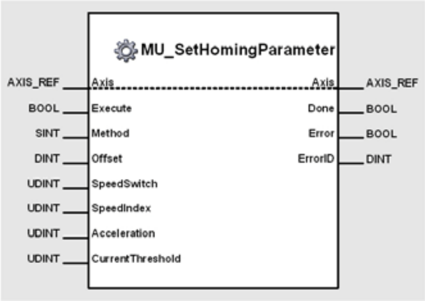


Figure 5-75 MU_SetHomingParameter

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	–
	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
	SpeedSwitch	UDINT	100	0…max. profile velocity	rpm
	SpeedIndex	UDINT	100	0…max. profile velocity	rpm
	Acceleration	UDINT	1000	0…max. acceleration	rpm/s
	CurrentThreshold	UINT	500	0 and up (depending on hardware)	mA
Output	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-142	–

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

Table 5-55 MU_SetHomingParameter

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 parameters *)
fbSetHomingParameter(Axis := myAxis, Execute := TRUE, Method :=11, Offset:= 200,
SpeedSwitch := 150, SpeedIndex := 20, Acceleration := 2000, CurrentThreshold := 500);
```

5.2.2 Position Mode

5.2.2.1 MU_ActivatePositionMode

Sets the «PositionMode» as active operation mode.

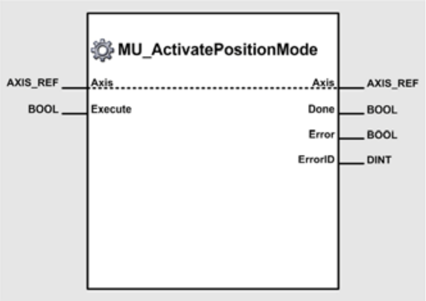


Figure 5-76 MU_ActivatePositionMode

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 8-142	–

I) A positive edge of *Execute* triggers the activation of position mode.
O) Successful operation is signalled with a positive value (TRUE) at *Done*.

Table 5-56 MU_ActivatePositionMode

Call

```
-------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbActivate : MU_ActivatePositionMode; (* fbActivate is instance of
MU_ActivatePositionMode *)
END_VAR
-------------------------------------------------------------------------------------
(* Function Block call for activating position mode*)
fbActivate (Axis := myAxis, Execute := TRUE);
```

5.2.2.2 MU_SetPositionMust

Sets the Position Mode setpoint.

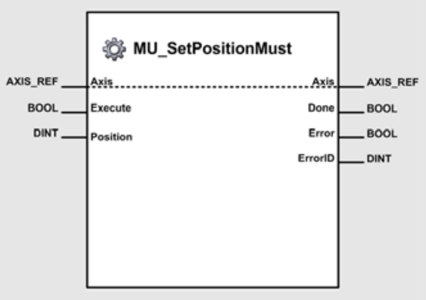


Figure 5-77 MU_SetPositionMust

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$^{*O)}$	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-142	–

I) A positive edge of *Execute* triggers a write operation of the position mode setting value object.
O) Successful operation is signalled with a positive value (TRUE) at *Done*.

Table 5-57 MU_SetPositionMust

Call

```
-------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbSet : MU_SetPositionMust; (* fbSet is instance of MU_SetPositionMust *)
END_VAR
-------------------------------------------------------------------------------------
(* Function Block call for writing position mode setting value *)
fbSet (Axis := myAxis, Execute := TRUE, Position := 1000);
```

5.2.2.3 MU_EnableAnalogPositionSetpoint

Activates the analog position setpoint.

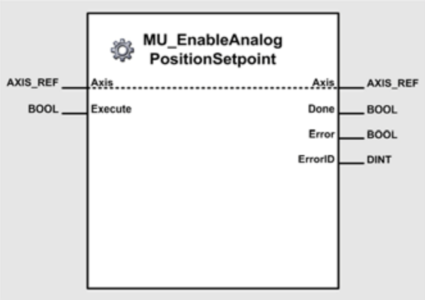


Figure 5-78 MU_EnableAnalogPositionSetpoint

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 8-142	–

I) A positive edge of *Execute* triggers activation of analog position setpoint.
O) Successful operation is signalled with a positive value (TRUE) at *Done*.

Table 5-58 MU_EnableAnalogPositionSetpoint

Call

```
------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbEnable : MU_EnableAnalogPositionSetpoint; (* fbEnable is instance of
MU_EnableAnalogPositionSetpoint *)
END_VAR
------------------------------------------------------------------------------------
(* Function Block call for analog position setpoint activation *)
fbEnable (Axis := myAxis, Execute := TRUE);
```

5.2.2.4 MU_DisableAnalogPositionSetpoint

Deactivates the analog position setpoint.

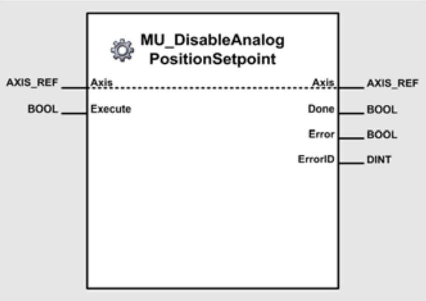


Figure 5-79 MU_DisableAnalogPositionSetpoint

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 8-142	–

I) A positive edge of *Execute* triggers deactivation of analog position setpoint.
O) Successful operation is signalled with a positive value (TRUE) at *Done*.

Table 5-59 MU_DisableAnalogPositionSetpoint

Call

```
------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbDisable : MU_DisableAnalogPositionSetpoint; (* fbDisable is instance of
MU_DisableAnalogPositionSetpoint *)
END_VAR
------------------------------------------------------------------------------------
(* Function Block call for analog position setpoint deactivation *)
fbDisable (Axis := myAxis, Execute := TRUE);
```

5.2.2.5 MU_GetAnalogPositionParameter

Reads the parameter for the analog position setpoint.

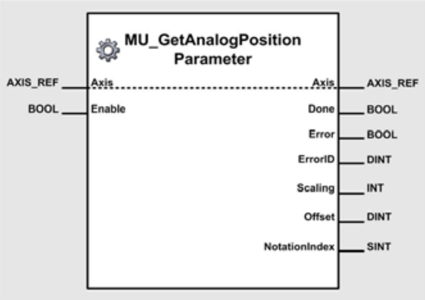


Figure 5-80 MU_GetAnalogPositionParameter

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 8-142	–
	Scaling	INT	0	-32‘767…+32‘768	qc/V
	Offset	DINT	0	-2‘147‘483‘648…+2‘147‘483‘647	qc
	NotationIndex	SINT	0	-2…0 (10^{-2}…10^{0}),	–

I) As long as *Enable* is TRUE (positive state), the values of the analog position setpoint objects are continuously being read.

O) The values of the objects can be read from *Scaling*, *Offset* and *NotationIndex*.

Table 5-60 MU_GetAnalogPositionParameter

Call

```
------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbGet : MU_GetAnalogPositionParameter; (* fbGet is instance of
MU_GetAnalogPositionParameter *)
END_VAR
------------------------------------------------------------------------------------
(* Function Block call for reading the analog position setpoint parameters *)
fbGet (Axis := myAxis, Enable := TRUE);
```

5.2.2.6 MU_SetAnalogPositionParameter

Writes the parameter for the analog position setpoint.

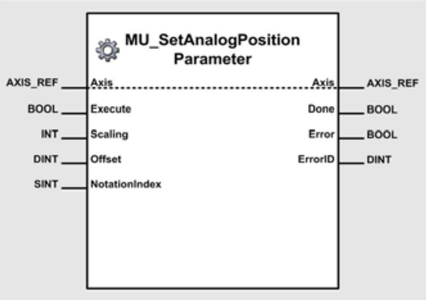


Figure 5-81 MU_SetAnalogPositionParameter

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	–
	Scaling	INT	0	-32‘767…+32‘768	qc/V
	Offset	DINT	0	-2‘147‘483‘648…+2‘147‘483‘647	qc
	NotationIndex	SINT	0	-2…0 (10^{-2}…10^{0}),	–
Output*O)	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes →page 8-142	–

I) A positive edge of *Execute* triggers a write operation of the analog position setpoint objects. *Scaling*, *Offset* and *NotationIndex* contain the value of the parameters to be written.

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

Table 5-61 MU_SetAnalogPositionParameter

Call

```
--------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbSet : MU_SetAnalogPositionParameter; (* fbSet is instance of
MU_SetAnalogPositionParameter *)
END_VAR
--------------------------------------------------------------------------------------
(* Function Block call for writing the analog position setpoint parameters *)
fbSet (Axis := myAxis, Execute := TRUE, Scaling := 0, Offset := 0, NotationIndex := 0);
```

5.2.3 Velocity Mode

5.2.3.1 MU_ActivateVelocityMode

Sets the «Velocity Mode» as active operation mode.

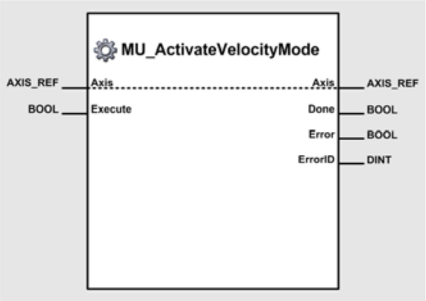


Figure 5-82 MU_ActivateVelocityMode

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 8-142	–

I) A positive edge of *Execute* triggers the activation of velocity mode.
O) Successful operation is signalled with a positive value (TRUE) at *Done*.

Table 5-62 MU_ActivateVelocityMode

Call

```
-------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbActivate : MU_ActivateVelocityMode; (* fbActivate is instance of
MU_ActivateVelocityMode *)
END_VAR
-------------------------------------------------------------------------------------
(* Function Block call for activating velocity mode*)
fbActivate (Axis := myAxis, Execute := TRUE);
```

5.2.3.2 MU_SetVelocityMust

Sets the Velocity Mode setpoint.

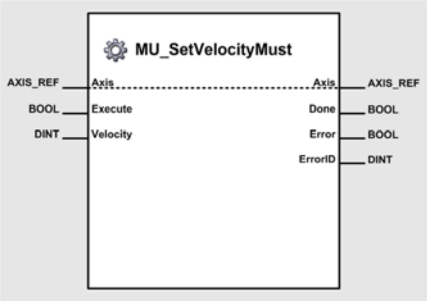


Figure 5-83 MU_SetPositionMust

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	–
	Velocity	DINT	0	±max. profile velocity	rpm
Output$^{*O)}$	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-142	–

I) A positive edge of *Execute* triggers a write operation of the velocity mode setting value object.
O) Successful operation is signalled with a positive value (TRUE) at *Done*.

Table 5-63 MU_SetPositionMust

Call

```
-------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbSet : MU_SetVelocityMust; (* fbSet is instance of MU_SetVelocityMust *)
END_VAR
-------------------------------------------------------------------------------------
(* Function Block call for writing velocity mode setting value *)
fbSet (Axis := myAxis, Execute := TRUE, Velocity := 100);
```

5.2.3.3 MU_EnableAnalogVelocitySetpoint

Activates the analog velocity setpoint.

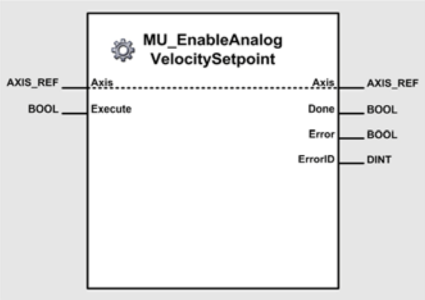


Figure 5-84 MU_EnableAnalogVelocitySetpoint

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 8-142	–

I) A positive edge of *Execute* triggers activation of analog velocity setpoint.
O) Successful operation is signalled with a positive value (TRUE) at *Done*.

Table 5-64 MU_EnableAnalogVelocitySetpoint

Call

```
-------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbEnable : MU_EnableAnalogVelocitySetpoint; (* fbEnable is instance of
MU_EnableAnalogVelocitySetpoint *)
END_VAR
-------------------------------------------------------------------------------------
(* Function Block call for analog velocity setpoint activation *)
fbEnable (Axis := myAxis, Execute := TRUE);
```

5.2.3.4 MU_DisableAnalogVelocitySetpoint

Deactivates the analog velocity setpoint.

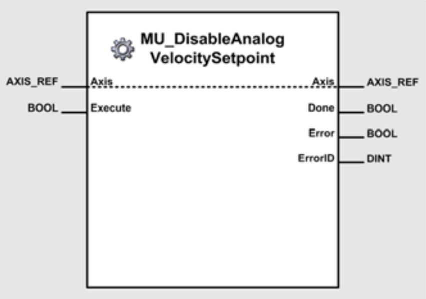


Figure 5-85 MU_DisableAnalogVelocitySetpoint

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 8-142	–

I) A positive edge of *Execute* triggers deactivation of analog velocity setpoint.

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

Table 5-65 MU_DisableAnalogVelocitySetpoint

Call

```
-------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbDisable : MU_DisableAnalogVelocitySetpoint; (* fbDisable is instance of
MU_DisableAnalogVelocitySetpoint *)
END_VAR
-------------------------------------------------------------------------------
(* Function Block call for analog velocity setpoint deactivation *)
fbDisable (Axis := myAxis, Execute := TRUE);
```

5.2.3.5 MU_GetAnalogVelocityParameter

Reads the parameter for the analog velocity setpoint.

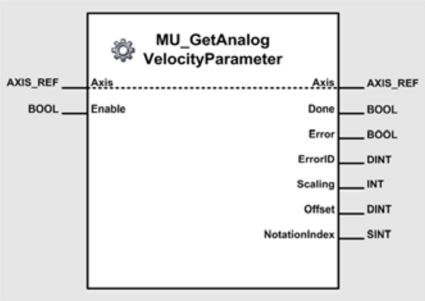


Figure 5-86 MU_GetAnalogVelocityParameter

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 8-142	–
	Scaling	INT	0	-32‘767…+32‘768	rpm/V
	Offset	SINT	0	±max. profile velocity	rpm
	NotationIndex	SINT	0	-2…0 (10^{-2}…10^{0}),	–

I) As long as *Enable* is TRUE (positive state), the values of the analog velocity setpoint objects are continuously being read.

O) The values of the objects can be read from *Scaling*, *Offset* and *NotationIndex*.

Table 5-66 MU_GetAnalogVelocityParameter

Call

```
------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbGet : MU_GetAnalogVelocityParameter; (* fbGet is instance of
MU_GetAnalogVelocityParameter *)
END_VAR
------------------------------------------------------------------------------------
(* Function Block call for reading the analog velocity setpoint parameters *)
fbGet (Axis := myAxis, Enable := TRUE);
```

5.2.3.6 MU_SetAnalogVelocityParameter

Writes the parameter for the analog velocity setpoint.

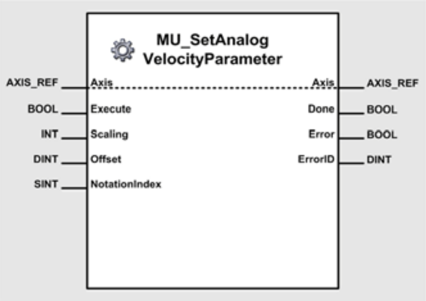


Figure 5-87 MU_SetAnalogVelocityParameter

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	–
	Scaling	INT	0	-32‘767…+32‘768	rpm/V
	Offset	SINT	0	±max. profile velocity	rpm
	NotationIndex	SINT	0	-2…0 $(10^{-2}\dots10^{0})$,	–
Output*O)	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes →page 8-142	–

I) A positive edge of *Execute* triggers a write operation of the analog velocity setpoint objects. *Scaling*, *Offset* and *NotationIndex* contain the value of the parameters to be written.

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

Table 5-67 MU_SetAnalogVelocityParameter

Call

```
-----------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbSet : MU_SetAnalogVelocityParameter; (* fbSet is instance of
MU_SetAnalogVelocityParameter *)
END_VAR
-----------------------------------------------------------------------------------
(* Function Block call for writing the analog velocity setpoint parameters *)
fbSet (Axis := myAxis, Execute := TRUE, Scaling := 0, Offset := 0, NotationIndex := 0);
```

5.2.4 Current Mode

5.2.4.1 MU_ActivateCurrentMode

Sets the «Current Mode» as active operation mode.

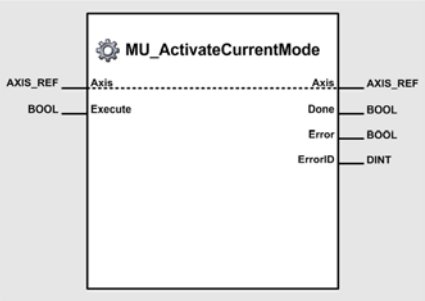


Figure 5-88 MU_ActivateCurrentMode

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 8-142	–

I) A positive edge of *Execute* triggers the activation of current mode.
O) Successful operation is signalled with a positive value (TRUE) at *Done*.

Table 5-68 MU_ActivateCurrentMode

Call

```
---------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbActivate : MU_ActivateCurrentMode; (* fbActivate is instance of
MU_ActivateCurrentMode *)
END_VAR
---------------------------------------------------------------------------------
(* Function Block call for activating current mode*)
fbActivate (Axis := myAxis, Execute := TRUE);
```

5.2.4.2 MU_SetCurrentMust

Sets the Current Mode setpoint.

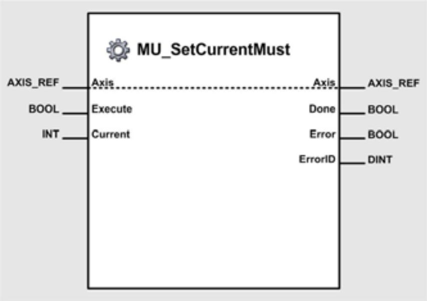


Figure 5-89 MU_SetCurrentMust

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	–
	Current	INT	0	depends on hardware	mA
Output*O)	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-142	–

I) A positive edge of *Execute* triggers a write operation of the current mode setting value object.
O) Successful operation is signalled with a positive value (TRUE) at *Done*.

Table 5-69 MU_SetCurrentMust

Call

```
-----------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbSet : MU_SetCurrentMust; (* fbSet is instance of MU_SetCurrentMust *)
END_VAR
-----------------------------------------------------------------------------------
(* Function Block call for writing current mode setting value *)
fbSet (Axis := myAxis, Execute := TRUE, Current := 100);
```

5.2.4.3 MU_EnableAnalogCurrentSetpoint

Activates the analog current setpoint.

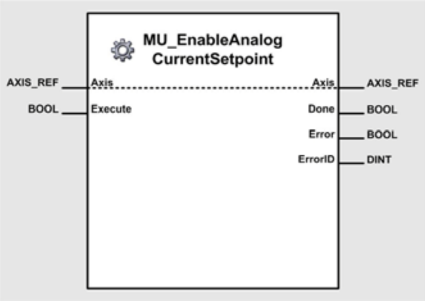


Figure 5-90 MU_EnableAnalogCurrentSetpoint

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 8-142	–

I) A positive edge of *Execute* triggers activation of analog current setpoint.
O) Successful operation is signalled with a positive value (TRUE) at *Done*.

Table 5-70 MU_EnableAnalogCurrentSetpoint

Call

```
-----------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbEnable : MU_EnableAnalogCurrentSetpoint; (* fbEnable is instance of
MU_EnableAnalogCurrentSetpoint *)
END_VAR
-----------------------------------------------------------------------------------
(* Function Block call for analog current setpoint activation *)
fbEnable (Axis := myAxis, Execute := TRUE);
```

5.2.4.4 MU_DisableAnalogCurrentSetpoint

Deactivates the analog current setpoint.

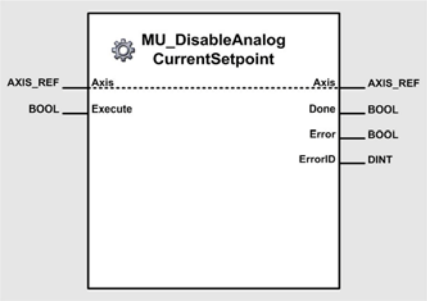


Figure 5-91 MU_DisableAnalogCurrentSetpoint

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 8-142	–

I) A positive edge of *Execute* triggers deactivation of analog current setpoint.
O) Successful operation is signalled with a positive value (TRUE) at *Done*.

Table 5-71 MU_DisableAnalogCurrentSetpoint

Call

```
---------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbDisable : MU_DisableAnalogCurrentSetpoint; (* fbDisable is instance of
MU_DisableAnalogCurrentSetpoint *)
END_VAR
---------------------------------------------------------------------------------
(* Function Block call for analog current setpoint deactivation *)
fbDisable (Axis := myAxis, Execute := TRUE);
```

5.2.4.5 MU_GetAnalogCurrentParameter

Reads the parameter for the analog current setpoint.

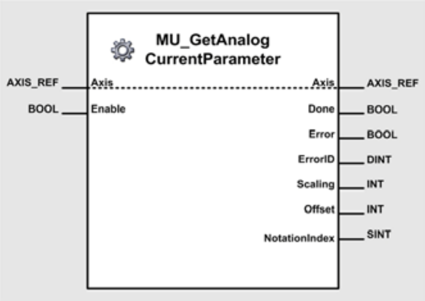


Figure 5-92 MU_GetAnalogCurrentParameter

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 8-142	–
	Scaling	INT	0	-32‘767…+32‘768	mA/V
	Offset	DINT	0	depends on hardware	mA
	NotationIndex	SINT	0	-2…0 (10^{-2}…10^{0}),	–

I) As long as *Enable* is TRUE (positive state), the values of the analog current setpoint objects are continuously being read.

O) The values of the objects can be read from *Scaling*, *Offset* and *NotationIndex*.

Table 5-72 MU_GetAnalogCurrentParameter

Call

```
--------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbGet : MU_GetAnalogCurrentParameter; (* fbGet is instance of
MU_GetAnalogCurrentParameter *)
END_VAR
--------------------------------------------------------------------------------
(* Function Block call for reading the analog current setpoint parameters *)
fbGet (Axis := myAxis, Enable := TRUE);
```

5.2.4.6 MU_SetAnalogCurrentParameter

Writes the parameter for the analog current setpoint.

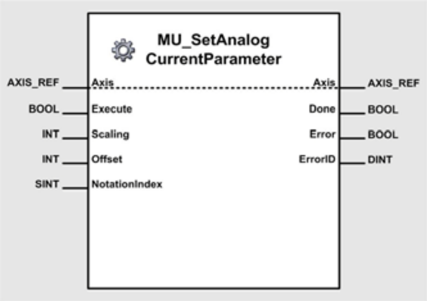


Figure 5-93 MU_SetAnalogCurrentParameter

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	–
	Scaling	INT	0	-32‘767…+32‘768	mA/V
	Offset	DINT	0	depends on hardware	mA
	NotationIndex	SINT	0	-2…0 (10^{-2}…10^{0}),	–
Output*O)	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes →page 8-142	–

I) A positive edge of *Execute* triggers a write operation of the analog current setpoint objects. *Scaling*, *Offset* and *NotationIndex* contain the value of the parameters to be written.

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

Table 5-73 MU_SetAnalogCurrentParameter

Call

```
-------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbSet : MU_SetAnalogCurrentParameter; (* fbSet is instance of
MU_SetAnalogCurrentParameter *)
END_VAR
-------------------------------------------------------------------------------------
(* Function Block call for writing the analog current setpoint parameters *)
fbSet (Axis := myAxis, Execute := TRUE, Scaling := 0, Offset := 0, NotationIndex := 0);
```

5.2.5 Master Encoder Mode

5.2.5.1 MU_ActivateMasterEncoderMode

Sets the «Master Encoder Mode» as active operation mode.

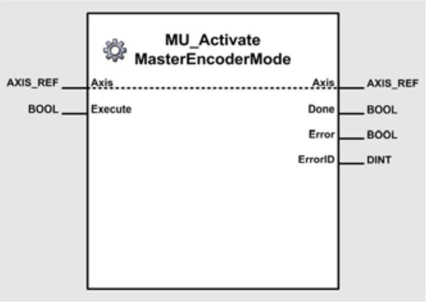


Figure 5-94 MU_ActivateMasterEncoderMode

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 8-142	–

I) A positive edge of *Execute* triggers the activation of master encoder mode.
O) Successful operation is signalled with a positive value (TRUE) at *Done*.

Table 5-74 MU_ActivateMasterEncoderMode

Call

```
-------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbAct : MU_ActivateMasterEncoderMode; (* fbAct is instance of
MU_ActivateMasterEncoderMode *)
END_VAR
-------------------------------------------------------------------------------
(* Function Block call for activating master encoder mode *)
fbAct (Axis := myAxis, Execute := TRUE);
```

5.2.5.2 MU_GetMasterEncoderParameter

Reads the Master Encoder Mode parameter.

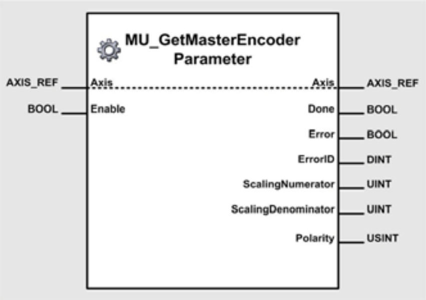


Figure 5-95 MU_GetMasterEncoderParameter

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 8-142	–
	ScalingNumerator	UINT	1	0…65’535	–
	ScalingDenominator	UINT	1	0…65’535	–
	Polarity	USINT	0	0, 1	–

I) As long as *Enable* is TRUE (positive state), the values of the master encoder mode objects are continuously being read.

O) The values of the objects can be read from *ScalingNumerator*, *ScalingDenominator* and *Polarity*.

Table 5-75 MU_GetMasterEncoderParameter

Call

```
-----------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbGet : MU_GetMasterEncoderParameter; (* fbGet is instance of
MU_GetMasterEncoderParameter *)
END_VAR
-----------------------------------------------------------------------------------
(* Function Block call for reading the master encoder mode parameters *)
fbGet (Axis := myAxis, Enable := TRUE);
```

5.2.5.3 MU_SetMasterEncoderParameter

Writes the Master Encoder Mode parameter.

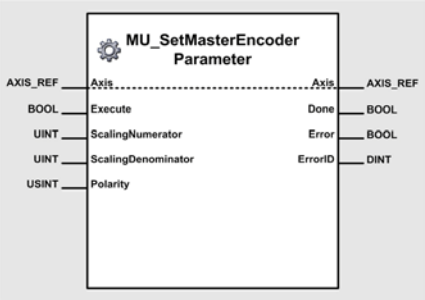


Figure 5-96 MU_SetMasterEncoderParameter

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	–
	ScalingNumerator	UINT	1	0…65’535	–
	ScalingDenominator	UINT	1	0…65’535	–
	Polarity	USINT	0	0, 1	–
Output*O)	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-142	–

I) A positive edge of *Execute* triggers a write operation of the master encoder mode objects. *ScalingNumerator*, *ScalingDenominator* and *Polarity* contain the value of the parameters to be written.

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

Table 5-76 MU_SetMasterEncoderParameter

Call

```
---------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbSet : MU_SetMasterEncoderParameter; (* fbSet is instance of
MU_SetMasterEncoderParameter *)
END_VAR
---------------------------------------------------------------------------------------
(* Function Block call for writing the master encoder mode parameters *)
fbSet (Axis := myAxis, Execute := TRUE, ScalingNumerator := 1, ScalingDenominator :=
1, Polarity := 0);
```

5.2.6 Step/Direction Mode

5.2.6.1 MU_ActivateStepDirectionMode

Sets the «Step/Direction Mode» as active operation mode.

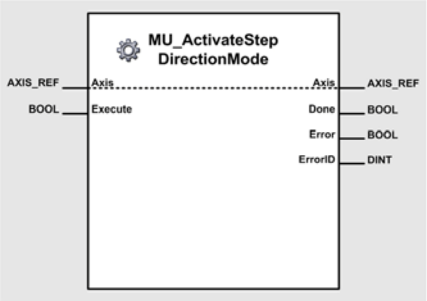


Figure 5-97 MU_ActivateStepDirectionMode

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 8-142	–

I) A positive edge of *Execute* triggers the activation of step direction mode.
O) Successful operation is signalled with a positive value (TRUE) at *Done*.

Table 5-77 MU_ActivateStepDirectionMode

Call

```
-----------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbAct : MU_ActivateStepDirectionMode; (* fbAct is instance of
MU_ActivateStepDirectionMode *)
END_VAR
-----------------------------------------------------------------------------------
(* Function Block call for activating step direction mode *)
fbAct (Axis := myAxis, Execute := TRUE);
```

5.2.6.2 MU_GetStepDirectionParameter

Reads the Step/Direction Mode parameter.

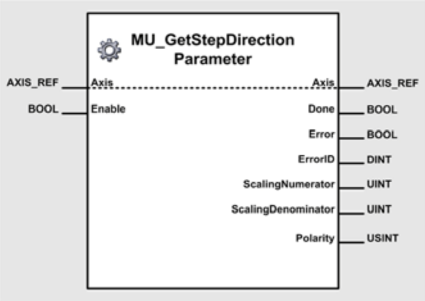


Figure 5-98 MU_GetStepDirectionParameter

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 8-142	–
	ScalingNumerator	UINT	1	0…65’535	–
	ScalingDenominator	UINT	1	0…65’535	–
	Polarity	USINT	0	0, 1	–

I) As long as *Enable* is TRUE (positive state), the values of the step direction mode objects are continuously being read.

O) The values of the objects can be read from *ScalingNumerator*, *ScalingDenominator* and *Polarity*.

Table 5-78 MU_GetStepDirectionParameter

Call

```
-----------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbGet : MU_GetStepDirectionParameter; (* fbGet is instance of
MU_GetStepDirectionParameter *)
END_VAR
-----------------------------------------------------------------------------------
(* Function Block call for reading the step direction mode parameters *)
fbGet (Axis := myAxis, Enable := TRUE);
```

5.2.6.3 MU_SetStepDirectionParameter

Writes the Step/Direction Mode parameter.

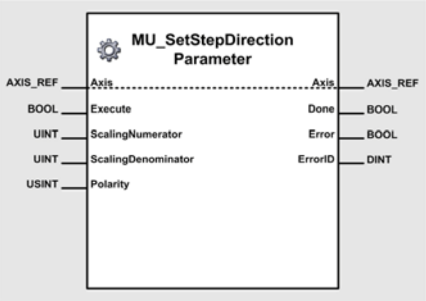


Figure 5-99 MU_SetStepDirectionParameter

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	–
	ScalingNumerator	UINT	1	0…65’535	–
	ScalingDenominator	UINT	1	0…65’535	–
	Polarity	USINT	0	0, 1	–
Output*O)	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-142	–

I) A positive edge of *Execute* triggers a write operation of the step direction mode objects. *ScalingNumerator*, *ScalingDenominator* and *Polarity* contain the value of the parameters to be written.

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

Table 5-79 MU_SetStepDirectionParameter

Call

```
-------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbSet : MU_SetStepDirectionParameter; (* fbSet is instance of
MU_SetStepDirectionParameter *)
END_VAR
-------------------------------------------------------------------------------------
(* Function Block call for writing the step direction mode parameters *)
fbSet (Axis := myAxis, Execute := TRUE, ScalingNumerator := 1, ScalingDenominator :=
1, Polarity := 0);
```

5.2.7 Interpolated Position Mode

5.2.7.1 MU_ActivateInterpolatedPositionMode

Sets the «Interpolated Position Mode» as active operation mode.

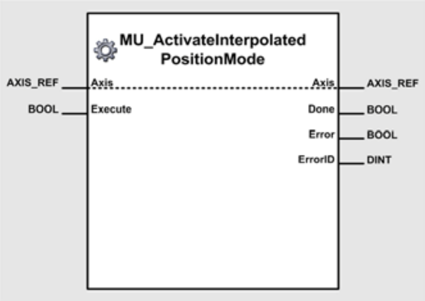


Figure 5-100 MU_ActivateInterpolatedPositionMode

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 8-142	–

I) A positive edge of *Execute* triggers the activation of interpolated position mode.
O) Successful operation is signalled with a positive value (TRUE) at *Done*.

Table 5-80 MU_ActivateInterpolatedPositionMode

Call

```
-----------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbAct : MU_ActivateInterpolatedPositionMode; (* fbAct is instance of
MU_ActivateInterpolatedPositionMode *)
END_VAR
-----------------------------------------------------------------------------------
(* Function Block call for activating interpolated position mode *)
fbAct (Axis := myAxis, Execute := TRUE);
```

5.2.7.2 MU_ClearIpmBuffer

Clears all PVT interpolation points from the IPM buffer.

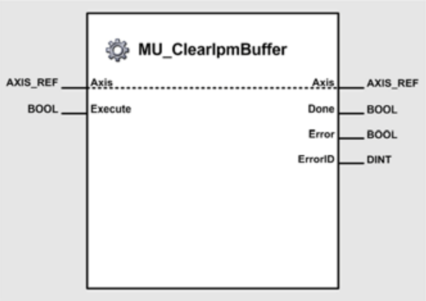


Figure 5-101 MU_ClearIpmBuffer

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 8-142	–

I) A positive edge of *Execute* clears the interpolated position mode buffer.
O) Successful operation is signalled with a positive value (TRUE) at *Done*.

Table 5-81 MU_ClearIpmBuffer

Call

```
-----------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbClear : MU_ClearIpmBuffer; (* fbClear is instance of MU_ClearIpmBuffer *)
END_VAR
-----------------------------------------------------------------------------------
(* Function Block call for clearing the interpolated position mode buffer*)
fbClear (Axis := myAxis, Execute := TRUE);
```

5.2.7.3 MU_AddPvtValues

Writes a PVT interpolation array to the IPM buffer.

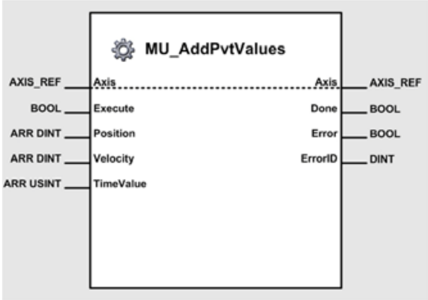


Figure 5-102 MU_AddPvtValues

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	ARRAY [1…64] OF DINT	–	-2'147'483'648…+2'147'483'647	qc
	Velocity	ARRAY [1…64] OF DINT	–	±max. profile velocity	rpm
	TimeValue	ARRAY [1…64] OF USINT	–	0…255	ms
Output*O)	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-142	–

I) A positive edge of *Execute* triggers a write operation of an array of PVT interpolation points to the IPM buffer.
Position, *Velocity* and *Time* contain the values to be written.

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

Table 5-82 MU_AddPvtValues

Call

```
-------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbAdd : MU_AddPvtValues; (* fbAdd is instance of MU_AddPvtValues *)
PositionValues : ARRAY [1 .. 64] OF DINT;
VelocityValues : ARRAY [1 .. 64] OF DINT;
TimeValues: ARRAY [1 .. 64] OF USINT;
END_VAR
-------------------------------------------------------------------------------------
(* Function Block call for writing a PVT interpolation point to the IPM buffer *)
fbAdd (Axis := myAxis, Execute := TRUE, Position := PositionValues, Velocity := Veloc-
ityValues,
TimeValue := TimeValues);
```

5.2.7.4 MU_AddPvtValue

Writes a PVT interpolation point to the IPM buffer.

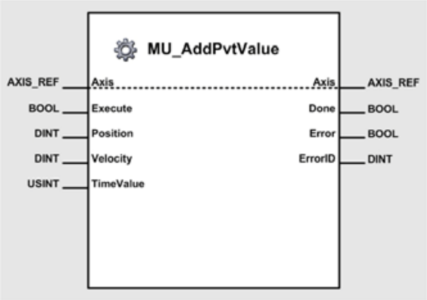


Figure 5-103 MU_AddPvtValue

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	–	-2‘147‘483‘648…+2‘147‘483‘647	qc
	Velocity	DINT	–	±max. profile velocity	rpm
	TimeValue	USINT	–	0…255	ms
Output$^{*O)}$	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-142	–

I) A positive edge of *Execute* triggers a write operation of a PVT interpolation point to the IPM buffer.
Position, *Velocity* and *Time* contain the values to be written.

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

Table 5-83 MU_AddPvtValue

Call

```
------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbAdd : MU_AddPvtValue; (* fbAdd is instance of MU_AddPvtValue *)
END_VAR
------------------------------------------------------------------------------------
(* Function Block call for writing a PVT interpolation point to the IPM buffer *)
fbAdd (Axis := myAxis, Execute := TRUE, Position := 1000, Velocity := 100, TimeValue :=
200);
```

5.2.7.5 MU_StartIpmTrajectory

Initiates an IPM trajectory.

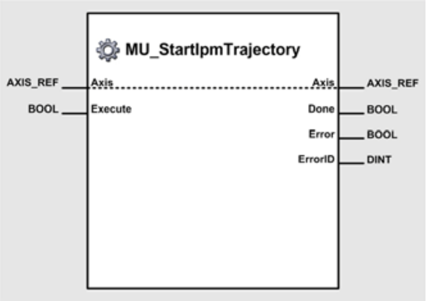


Figure 5-104 MU_StartIpmTrajectory

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 8-142	–

I) A positive edge of *Execute* starts the interpolated position mode trajectory.
O) Successful operation is signalled with a positive value (TRUE) at *Done*.

Table 5-84 MU_StartIpmTrajectory

Call

```
--------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbStart : MU_StartIpmTrajectory; (* fbStart is instance of MU_StartIpmTrajectory *)
END_VAR
--------------------------------------------------------------------------------------
(* Function Block call for starting the interpolated position mode trajectory*)
fbStart (Axis := myAxis, Execute := TRUE);
```

5.2.7.6 MU_StopIpmTrajectory

Stops an IPM trajectory.

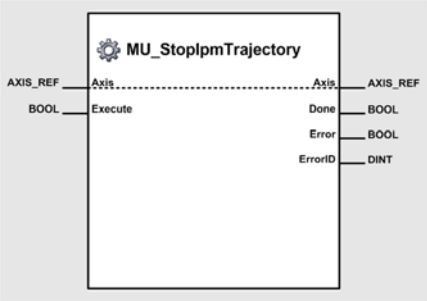


Figure 5-105 MU_StopIpmTrajectory

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 8-142	–

I) A positive edge of *Execute* stops the interpolated position mode trajectory.
O) Successful operation is signalled with a positive value (TRUE) at *Done*.

Table 5-85 MU_StopIpmTrajectory

Call

```
-----------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbStop : MU_StopIpmTrajectory; (* fbStop is instance of MU_StopIpmTrajectory *)
END_VAR
-----------------------------------------------------------------------------------
(* Function Block call for stopping the interpolated position mode trajectory*)
fbStop (Axis := myAxis, Execute := TRUE);
```

5.2.7.7 MU_GetIpmStatus

Reads the IPM status.

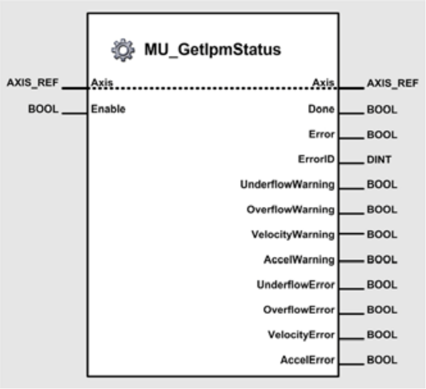


Figure 5-106 MU_GetIpmStatus

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 8-142	–
	UnderflowWarning	BOOL	FALSE	TRUE, FALSE	–
	OverflowWarning	BOOL	FALSE	TRUE, FALSE	–
	VelocityWarning	BOOL	FALSE	TRUE, FALSE	–
	AccelWarning	BOOL	FALSE	TRUE, FALSE	–
	UnderflowError	BOOL	FALSE	TRUE, FALSE	–
	OverflowError	BOOL	FALSE	TRUE, FALSE	–
	VelocityError	BOOL	FALSE	TRUE, FALSE	–
	AccelError	BOOL	FALSE	TRUE, FALSE	–

I) As long as *Enable* is TRUE (positive state), the values of the interpolated position mode status are continuously being read.

O) The values can be read from *UnderflowWarning*, *OverflowWarning*, *VelocityWarning*, *AccelWarning*, *UnderflowError*, *OverflowError*, *VelocityError* and *AccelError*.

Table 5-86 MU_GetIpmStatus

Call

```
-----------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbGet : MU_GetIpmStatus; (* fbGet is instance of MU_GetIpmStatus *)
END_VAR
-----------------------------------------------------------------------------------
(* Function Block call for reading the interpolated position mode status *)
fbGet (Axis := myAxis, Enable := TRUE);
```

5.2.7.8 MU_GetIpmTrajectoryStatus

Reads the status of the IPM trajectory.

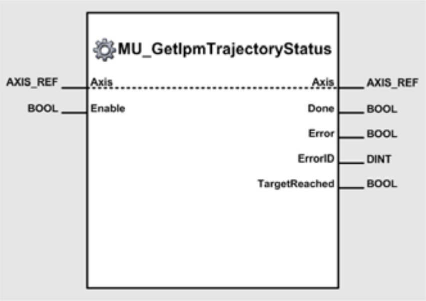


Figure 5-107 MU_GetIpmTrajectoryStatus

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 8-142	–
	TargetReached	BOOL	FALSE	TRUE, FALSE	–

I) As long as *Enable* is TRUE (positive state), the values of the trajectory status are continuously being read.

O) The status values can be read from *TargetReached*.

Table 5-87 MU_GetIpmTrajectoryStatus

Call

```
-----------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbGet : MU_GetIpmTrajectoryStatus; (* fbGet is instance of MU_GetIpmTrajectoryStatus
*)
END_VAR
-----------------------------------------------------------------------------------
(* Function Block call for reading the interpolated position trajectory status *)
fbGet (Axis := myAxis, Enable := TRUE);
```

5.2.8 Inputs and Outputs

5.2.8.1 MU_GetAllDigitalInputs

Returns the state of all digital inputs.

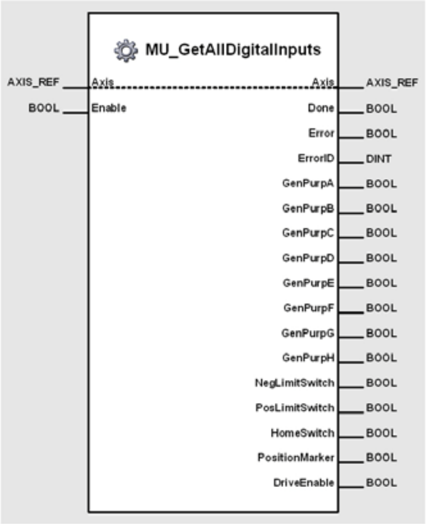


Figure 5-108 MU_GetAllDigitalInputs

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 8-142	–
	GenPurpA	BOOL	FALSE	TRUE, FALSE	–
	GenPurpB	BOOL	FALSE	TRUE, FALSE	–
	GenPurpC	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	–
	NegLimitSwitch	BOOL	FALSE	TRUE, FALSE	–
	PosLimitSwitch	BOOL	FALSE	TRUE, FALSE	–
	HomeSwitch	BOOL	FALSE	TRUE, FALSE	–
	PositionMarker	BOOL	FALSE	TRUE, FALSE	–
	DriveEnable	BOOL	FALSE	TRUE, FALSE	–

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

O The values of the objects can be read from *GenPurpA*, …, *DriveEnable*.

Table 5-88 MU_GetAllDigitalInputs

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);
```

5.2.8.2 MU_GetDigitalInput

Returns the state of a specific digital input.

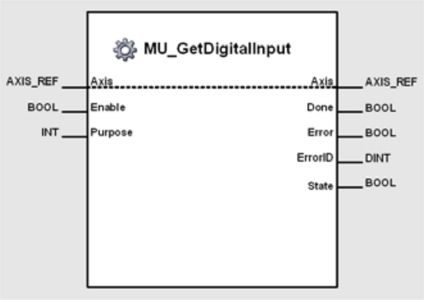


Figure 5-109 MU_GetDigitalInput

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[*O)]	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-142	–
	State	BOOL	FALSE	TRUE, FALSE	–

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

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

Table 5-89 MU_GetDigitalInput

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);
```

5.2.8.3 MU_GetAnalogInput

Returns the value of a specific analog input.

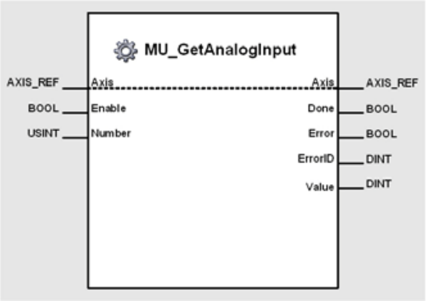


Figure 5-110 MU_GetAnalogInput

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*O)	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-142	–
	Value	DINT	0	0…5’000	mV

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

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

Table 5-90 MU_GetAnalogInput

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);
```

5.2.8.4 MU_SetAllDigitalOutputs

Modifies the value of all digital outputs.

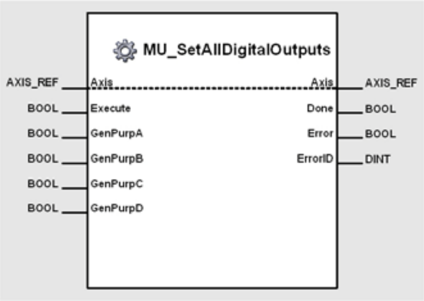


Figure 5-111 MU_SetAllDigitalOutputs

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*O)	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes →page 8-142	–

I) A positive edge of *Execute* triggers a write operation of all digital outputs.
O) Successful operation is signalled with a positive value (TRUE) at *Done*.

Table 5-91 MU_SetAllDigitalOutputs

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 *)
fbSetAllDigitalOutputs(Axis := myAxis, Execute := TRUE, GenPurpA := TRUE, GenPurpB :=
TRUE, GenPurpC := TRUE, GenPurpD := TRUE);
```

5.2.9 Position Marker

5.2.9.1 MU_ReadPositionMarkerCounter

Reads number of recorded position markers.

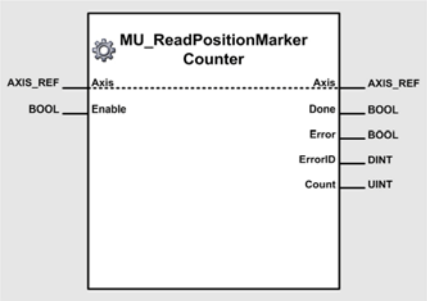


Figure 5-112 MU_ReadPositionMarkerCounter

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 8-142	–
	Count	UINT	0	0…3	–

I) As long as *Enable* is TRUE (positive state), the value of the position marker counter is continuously being read.

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

Table 5-92 MU_ReadPositionMarkerCounter

Call

```
---------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbRead : MU_ReadPositionMarkerCounter; (* fbRead is instance of
MU_ReadPositionMarkerCounter *)
END_VAR
---------------------------------------------------------------------------------------
(* Function Block call for reading the position marker counter *)
fbRead (Axis := myAxis, Enable := TRUE);
```

5.2.9.2 MU_ReadCapturedPosition

Reads a recorded position marker.

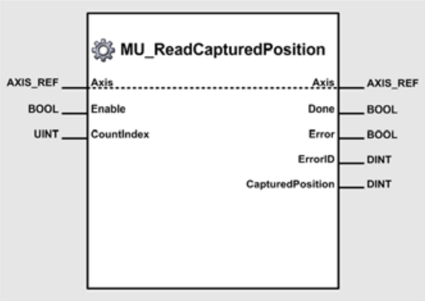


Figure 5-113 MU_ReadCapturedPosition

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	–
	CountIndex	UINT	0	0…Count-1 (➔page 5-115)	–
Output*O)	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-142	–
	CapturedPosition	DINT	0	-2‘147‘483‘648…+2‘147‘483‘647	qc

I) As long as *Enable* is TRUE (positive state), the value of the captured position is continuously being read.

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

Table 5-93 MU_ReadCapturedPosition

Call

```
---------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbRead : MU_ReadCapturedPosition; (* fbRead is instance of MU_ReadCapturedPosition *)
END_VAR
---------------------------------------------------------------------------------------
(* Function Block call for reading the captured position *)
fbRead (Axis := myAxis, Enable := TRUE, CountIndex: = 1);
```

5.2.9.3 MU_ResetCapturedPosition

Resets a recorded position marker.

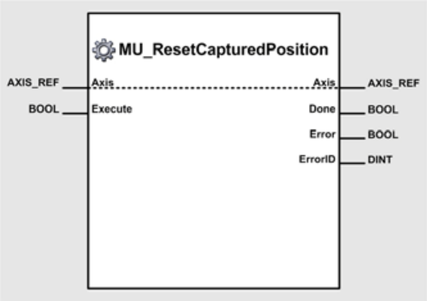


Figure 5-114 MU_ResetCapturedPosition

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 8-142	–

I) A positive edge of *Execute* triggers a reset of the captured position.
O) Successful operation is signalled with a positive value (TRUE) at *Done*.

Table 5-94 MU_ResetCapturedPosition

Call

```
-----------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbReset : MU_ResetCapturedPosition; (* fbReset is instance of MU_ResetCapturedPosition
*)
END_VAR
-----------------------------------------------------------------------------------
(* Function Block call for a reset of the captured position *)
fbReset (Axis := myAxis, Execute := TRUE);
```

5.2.10 Position Compare

5.2.10.1 MU_EnablePositionCompare

Activates the «Position Compare» function.

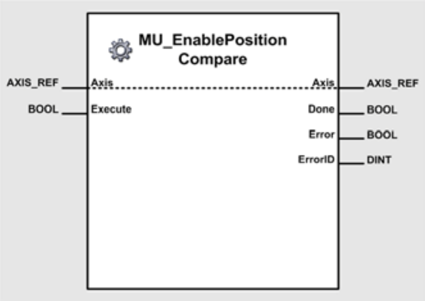


Figure 5-115 MU_EnablePositionCompare

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 8-142	–

I) A positive edge of *Execute* triggers activation of position compare functionality.
O) Successful operation is signalled with a positive value (TRUE) at *Done*.

Table 5-95 MU_EnablePositionCompare

Call

```
-----------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbEnable : MU_EnablePositionCompare; (* fbEnable is instance of
MU_EnablePositionCompare *)
END_VAR
-----------------------------------------------------------------------------------
(* Function Block call for position compare activation *)
fbEnable (Axis := myAxis, Execute := TRUE);
```

5.2.10.2 MU_DisablePositionCompare

Deactivates the «Position Compare» function.

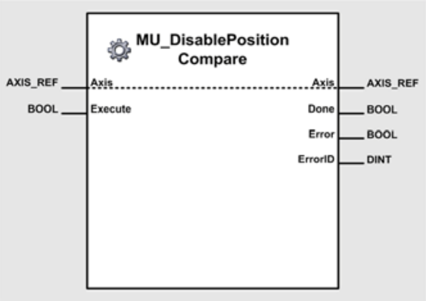


Figure 5-116 MU_DisablePositionCompare

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 8-142	–

I) A positive edge of *Execute* triggers deactivation of position compare functionality.

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

Table 5-96 MU_DisablePositionCompare

Call

```
-------------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbDisable : MU_DisablePositionCompare; (* fbDisable is instance of
MU_DisablePositonCompare *)
END_VAR
-------------------------------------------------------------------------------------
(* Function Block call for position compare deactivation *)
fbDisable (Axis := myAxis, Execute := TRUE);
```

5.2.10.3 MU_SetPositionCompareRefPos

Sets the reference position for the «Position Compare» function.

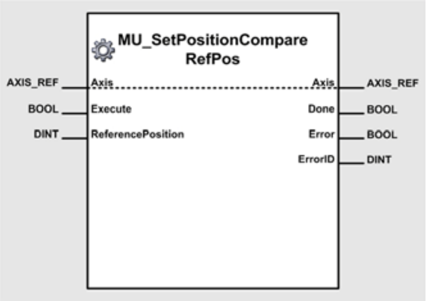


Figure 5-117 MU_SetPositionCompareRefPos

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	–
	ReferencePosition	DINT	0	-2‘147‘483‘648…+2‘147‘483‘647	qc
Output*O)	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-142	–

I) A positive edge of *Execute* triggers a write operation of the position compare reference position. *ReferencePosition* contains the value to be written.

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

Table 5-97 MU_SetPositionCompareRefPos

Call

```
-----------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
myAxis : AXIS_REF := (AxisNo := 0);
fbSet : MU_SetPositionCompareRefPos; (* fbSet is instance of
MU_SetPositionCompareRefPos *)
END_VAR
-----------------------------------------------------------------------------------
(* Function Block call for writing the position compare reference position*)
fbSet (Axis := myAxis, Execute := TRUE, ReferencePosition := 1000);
```

5.2.11 Error Handling

5.2.11.1 MU_GetDeviceErrorCount

Returns the number of actual errors.

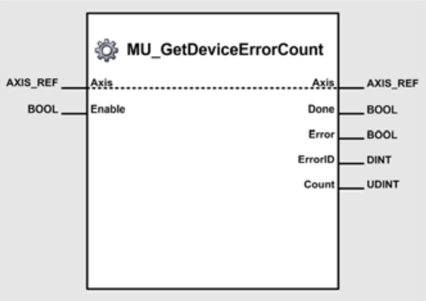


Figure 5-118 MU_GetDeviceErrorCount

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 8-142	–
	Count	UDINT	0	0…255	–

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

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

Table 5-98 MU_GetDeviceErrorCount

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);
```

5.2.11.2 MU_GetDeviceError

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

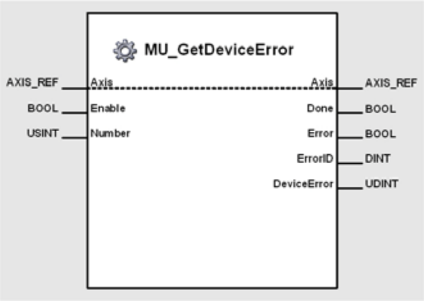


Figure 5-119 MU_GetDeviceError

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 5-121)	–
Output*O)	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-142	–
	DeviceError	UDINT	0	➔separate document «EPOS2 Firmware Specification»	–

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

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

Table 5-99 MU_GetDeviceError

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);
```

5.2.12 Object Access

5.2.12.1 MU_GetObject

Returns the value of an EPOS object.

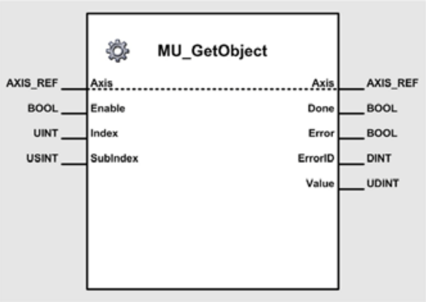


Figure 5-120 MU_GetObject

Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input/Output	Axis	AXIS_REF	0	0…31	AxisNo [USINT]
Input$^{*\mathrm{I})}$	Enable	BOOL	FALSE	TRUE, FALSE	–
	Index	UINT	0	➔separate document «EPOS2 Firmware Specification»	–
	SubIndex	USINT	0	➔separate document «EPOS2 Firmware Specification»	–
Output$^{*\mathrm{O})}$	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-142	–
	Value	UDINT	0	0…4’294’967’265	–

I) As long as *Enable* is TRUE (positive state), the values of the EPOS homing objects are continuously being read.
Index and *SubIndex* define the object to be read.

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

Table 5-100 MU_GetObject

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, Enable := TRUE, Index := 16#2003, SubIndex := 16#01);
```

5.2.12.2 MU_SetObject

Modifies the value of an EPOS object.

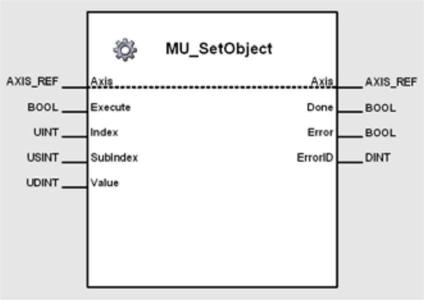


Figure 5-121 MU_SetObject

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	➔separate document «EPOS2 Firmware Specification»	–
	SubIndex	USINT	0	➔separate document «EPOS2 Firmware Specification»	–
	Value	UDINT	0	0…4’294’967’265	–
Output$^{*O)}$	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-142	–

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

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

Table 5-101 MU_SetObject

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,
Value :=512);
```

5.2.13 Data Handling

5.2.13.1 MU_Selection

Selects between two values.

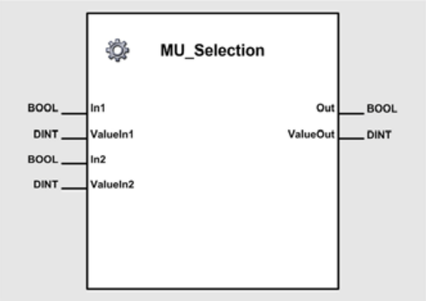


Figure 5-122 MU_Selection

Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input[*I)]	In1	BOOL	FALSE	TRUE, FALSE	–
	ValueIn1	DINT	0	-2‘147‘483‘648…+2‘147‘483‘647	–
	In2	BOOL	FALSE	TRUE, FALSE	–
	ValueIn2	DINT	0	-2‘147‘483‘648…+2‘147‘483‘647	–
Output[*O)]	Out	BOOL	FALSE	TRUE, FALSE	–
	ValueOut	DINT	0	-2‘147‘483‘648…+2‘147‘483‘647	–

I) *In1* selects *ValueIn1*, *In2* selects *ValueIn2*. If *In1* and *In2* are TRUE, *In1* is prioritized.
O) *Out* indicates a valid value of *ValueOut*.

Table 5-102 MU_Selection

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, ValueIn1 := 2000, In2 := FALSE, ValueIn2 := 1000);
```

5.2.13.2 MU_GetBitState

Extracts the state of a specific bit.

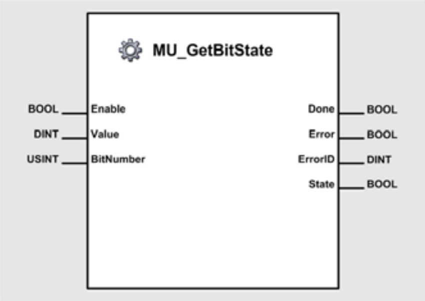


Figure 5-123 MU_GetBitState

Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input$^{*I)}$	Enable	BOOL	FALSE	TRUE, FALSE	–
	Value	DINT	0	-2‘147‘483‘648…+2‘147‘483‘647	qc
	BitNumber	USINT	0	0…31	–
Output$^{*O)}$	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-142	–
	State	BOOL	FALSE	TRUE, FALSE	–

I) As long as *Enable* is TRUE (positive state), the state of a specific bit within *Value* is continuously being read.

O) The state can be read from *State*.

Table 5-103 MU_GetBitState

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*)
```

5.2.13.3 MU_SetBitState

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

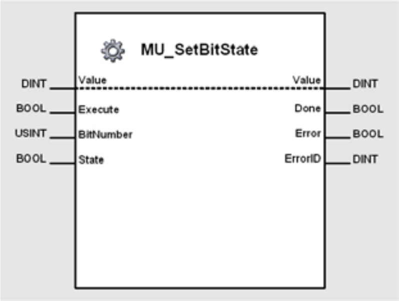


Figure 5-124 MU_SetBitState

Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input/Output	Value	DINT	0	-2‘147‘483‘648…+2‘147‘483‘647	–
Input*I)	Execute	BOOL	FALSE	TRUE, FALSE	–
	BitNumber	USINT	0	0…31	–
	State	BOOL	FALSE	TRUE, FALSE	–
Output*O)	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-142	–

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*.

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

Table 5-104 MU_SetBitState

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*)
```

5.2.13.4 MU_DataRecorder

Records data cyclic into a ring buffer.

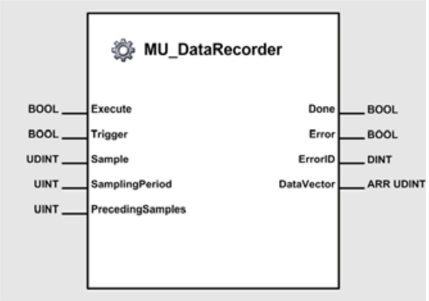


Figure 5-125 MU_DataRecorder

Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input*I)	Execute	BOOL	FALSE	TRUE, FALSE RisingE = Start FallingE = Stop	–
	Trigger	BOOL	FALSE	TRUE, FALSE	–
	Sample	UDINT		0…4‘294‘967‘295	–
	SamplingPeriod	UINT		0…65‘535	[cycle]
	PrecedingSamples	UINT		0…65‘535	[sample]
Output*O)	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-142	–
	DataVector	ARRAY [1…1000] OF UDINT		0…4‘294‘967‘295	–

I) A positive edge of *Execute* starts the data recorder, a negative edge of *Execute* stops the data recorder immediately.
A positive edge of *Trigger* triggers an event to stop the data recorder, but recording will be continued until the buffer is full.
Sample contains the value to be recorded.
SamplingPeriod determines the sampling rate as a factor of a program cycle.
PrecedingSamples determines the number of samples in the output data vector before the trigger event.

O) After a positive value (TRUE) at *Done*, the recorded data is available in *DataVector*.

Table 5-105 MU_DataRecorder

Call

```
------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
fbRecord : MU_DataRecorder; (* fbRecord is instance of MU_DataRecorder *)
END_VAR
------------------------------------------------------------------------------
(* Function Block call for recording samples *)
fbRecord (Axis := myAxis, Execute := TRUE, Sample := VariableX, SamplingPeriod := 1);
```

5.3 CANopen CiA 301 Function Blocks

5.3.1 CAN_Nmt

Permits change of network management state of a CANopen device.

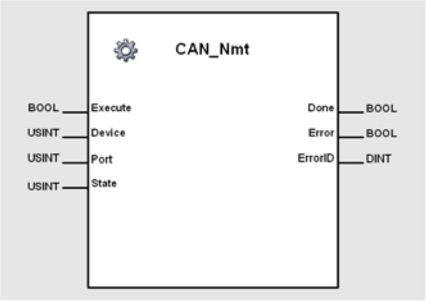


Figure 5-126 CAN_Nmt

Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input*I)	Execute	BOOL	FALSE	TRUE, FALSE	–
	Device	USINT	0	0…127	–
	Port	USINT	1	1 = CAN-I 2 = CAN-S	–
	State	USINT	0	1 = Start Remote Node 2 = Stop Remote Node 128 = Enter Pre-Operational 129 = Reset Node 130 = Reset Communication	–
Output*O)	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-142	–

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 Network (CAN-I) and Slave Network (CAN-S).
State is define by CANopen (➔CANopen specification).

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

Table 5-106 CAN_Nmt

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 := 2, State := 1);
```

5.3.2 CAN_SdoRead

Permits reading of a CANopen object using the SDO protocol.

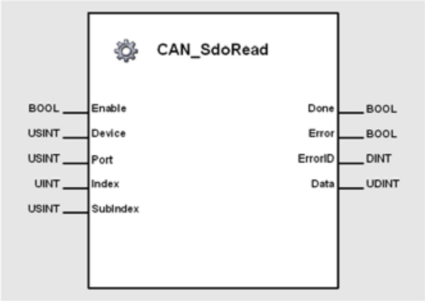


Figure 5-127 CAN_SdoRead

Important!
Execution of the instance might take longer than one PLC cycle (➔page 5-47).

Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input[*I)]	Enable	BOOL	FALSE	TRUE, FALSE	–
	Device	USINT	0	0…127 (Node ID)	–
	Port	USINT	1	1 = CAN-I 2 = CAN-S	–
	Index	UINT	0	➔separate document «EPOS2 Firmware Specification»	–
	SubIndex	USINT	0	➔separate document «EPOS2 Firmware Specification»	–
Output[*O)]	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-142	–
	Data	UDINT	0	0…4’294’967’295	–

I) As long as *Enable* is TRUE (positive state), the value of a specified CANopen object is continuously being read.
The object is specified by *Index* and *SubIndex*.
Device corresponds to the CAN Node-ID.
Port distinguishes between Internal Network (CAN-I) and Slave Network (CAN-S).

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

Table 5-107 CAN_SdoRead

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 := 1, Port := 0, Index := 16#1000, SubIndex :=
16#00);
```

5.3.3 CAN_SdoWrite

Permits writing of a CANopen object using the SDO protocol.

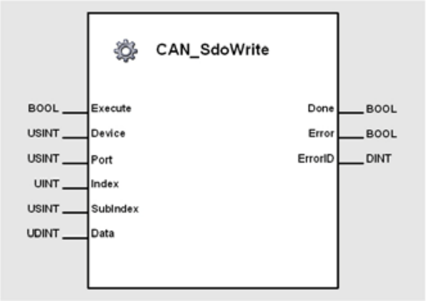


Figure 5-128 CAN_SdoWrite

Important!
Execution of the instance might take longer than one PLC cycle (➔page 5-47).

Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input[*I)]	Execute	BOOL	FALSE	TRUE, FALSE	–
	Device	USINT	0	0…127 (Node ID)	–
	Port	USINT	1	1 = CAN-I 2 = CAN-S	–
	Index	UINT	0	➔separate document «EPOS2 Firmware Specification»	–
	SubIndex	USINT	0	➔separate document «EPOS2 Firmware Specification»	–
	Data	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 8-142	–

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 Network (CAN-I) and Slave Network (CAN-S).

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

Table 5-108 CAN_SdoWrite

Call

```
------------------------------------------------------------------------------
(* Variable Declaration *)
(* 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 := 1, Port := 0, Index := 16#100C, SubIndex :=
16#00, Data := 100);
```

5.3.4 CAN_SetTxPdoEvent

Triggers a PDO of transmission type 254.

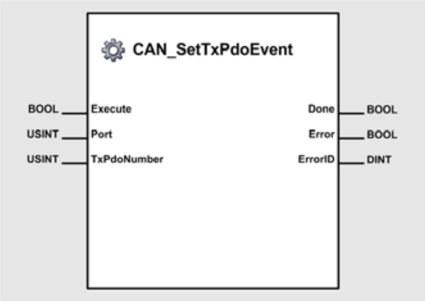


Figure 5-129 CAN_SetTxPdoEvent

Variables

Variable	Name	Data Type	Value		Unit –or– Element [Type]
			Default	Range	
Input *I)	Execute	BOOL	FALSE	TRUE, FALSE	–
	Port	USINT	1	1 = CAN-I 2 = CAN-S	–
	TxPdoNumber	USINT	1…4 1…32	CAN-I: PDO1…PDO4 CAN-S: PDO1…PDO32	–
Output *O)	Done	BOOL	FALSE	TRUE, FALSE	–
	Error	BOOL	FALSE	TRUE, FALSE	–
	ErrorID	DINT	0	For codes ➔page 8-142	–

I) A positive edge of *Execute* triggers a transmission of a TxPDO specified by TxPdoNumber. *Port* distinguishes between Internal Network (CAN-I) and Slave Network (CAN-S). *TxPdoNumber* defines number of PDO.

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

Table 5-109 CAN_SetTxPdoEvent

Call

```
--------------------------------------------------------------------------------
(* Variable Declaration *)
VAR
fbSetEvent : MC_SetTxPdoEvent; (* fbSetEvent is instance of MC_SetTxPdoEvent *)
END_VAR
--------------------------------------------------------------------------------
(* Function Block call for triggering a TxPDO1 transmission on port 2*)
fbSetEvent (Execute := TRUE, Port := 2, TxPdoNumber := 1);
```

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.

EPOS2 P uses 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 6-110 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 EPOS2 P global status register and is identical to the EPOS2 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 6-111 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 6-112 Global Status Register Markers – Examples

6.3 Marker Global Axis Error Register

Length is 32-bit. It holds the EPOS2 P global status register and is identical to the EPOS2 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 6-113 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 6-114 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 6-115 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 6-116 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 5-47).*

7.1 Internal Network

7.1.1 Process Inputs

Direct Input Variables Internal Network (Communication)

Quantity	Type	Address (BUS 1)	Description
4	SINT	AT %IB 1.0.0.0 - 1.0.3.0	Signed 8-Bit variable
4	USINT	AT %IB 1.1.0.0 - 1.1.3.0	Unsigned 8-Bit variable
4	INT	AT %IW 1.2.0.0 - 1.2.6.0	Signed 16-Bit variable
4	UINT	AT %IW 1.3.0.0 - 1.3.6.0	Unsigned 16-Bit variable
4	DINT	AT %ID 1.4.0.0 - 1.4.12.0	Signed 32-Bit variable
4	UDINT	AT %ID 1.5.0.0 - 1.5.12.0	Unsigned 32-Bit variable
2	LINT	AT %IL 1.6.0.0 - 1.6.8.0	Signed 64-Bit variable
2	ULINT	AT %IL 1.7.0.0 - 1.7.8.0	Unsigned 64-Bit variable

Table 7-117 Input Network Variables (IEC-61131 Program)

Process Input Objects Internal Network

Quantity	Type	Index, Subindex	Description
4	INT8	0xA000, 0x01…0x10	Signed 8-Bit object
4	UINT8	0xA040, 0x01…0x10	Unsigned 8-Bit object
4	INT16	0xA0C0, 0x01…0x10	Signed 16-Bit object
4	UINT16	0xA100, 0x01…0x10	Unsigned 16-Bit object
4	INT32	0xA1C0, 0x01…0x10	Signed 32-Bit object
4	UINT32	0xA200, 0x01…0x10	Unsigned 32-Bit object
2	INT64	0xA400, 0x01…0x10	Signed 64-Bit object
2	UINT64	0xA440, 0x01…0x10	Unsigned 64-Bit object

Table 7-118 Process Input Objects (Object Dictionary)

7.1.2 Process Outputs

Direct Output Variables Internal Network (Communication)

Quantity	Type	Address (BUS 1)	Description
4	SINT	AT %QB 1.0.0.0 - 1.0.3.0	Signed 8-Bit variable
4	USINT	AT %QB 1.1.0.0 - 1.1.3.0	Unsigned 8-Bit variable
4	INT	AT %QW 1.2.0.0 - 1.2.6.0	Signed 16-Bit variable
4	UINT	AT %QW 1.3.0.0 - 1.3.6.0	Unsigned 16-Bit variable
4	DINT	AT %QD 1.4.0.0 - 1.4.12.0	Signed 32-Bit variable
4	UDINT	AT %QD 1.5.0.0 - 1.5.12.0	Unsigned 32-Bit variable
2	LINT	AT %QL 1.6.0.0 - 1.6.8.0	Signed 64-Bit variable
2	ULINT	AT %QL 1.7.0.0 - 1.7.8.0	Unsigned 64-Bit variable

Table 7-119 Output Network Variables (IEC-61131 Program)

Process Output Objects Internal Network

Quantity	Type	Index, Subindex	Description
4	INT8	0xA480, 0x01…0x10	Signed 8-Bit object
4	UINT8	0xA4C0, 0x01…0x10	Unsigned 8-Bit object
4	INT16	0xA540, 0x01…0x10	Signed 16-Bit object
4	UINT16	0xA580, 0x01…0x10	Unsigned 16-Bit object
4	INT32	0xA640, 0x01…0x10	Signed 32-Bit object
4	UINT32	0xA680, 0x01…0x10	Unsigned 32-Bit object
2	INT64	0xA880, 0x01…0x10	Signed 64-Bit object
2	UINT64	0xA8C0, 0x01…0x10	Unsigned 64-Bit object

Table 7-120 Process Output Objects (Object Dictionary)

7.2 Slave Network

7.2.1 Process Inputs

Direct Input Variables Slave Network (Communication)

Quantity	Type	Address (BUS 2)	Description
64	SINT	AT %IB 2.0.0.0 - 2.0.63.0	Signed 8-Bit variable
64	USINT	AT %IB 2.1.0.0 - 2.1.63.0	Unsigned 8-Bit variable
64	INT	AT %IW 2.2.0.0 - 2.2.126.0	Signed 16-Bit variable
64	UINT	AT %IW 2.3.0.0 - 2.3.126.0	Unsigned 16-Bit variable
64	DINT	AT %ID 2.4.0.0 - 2.4.252.0	Signed 32-Bit variable
64	UDINT	AT %ID 2.5.0.0 - 2.5.252.0	Unsigned 32-Bit variable
32	LINT	AT %IL 2.6.0.0 - 2.6.248.0	Signed 64-Bit variable
32	ULINT	AT %IL 2.7.0.0 - 2.7.248.0	Unsigned 64-Bit variable

Table 7-121 Input Network Variables (IEC-61131 Program)

Process Input Objects Slave Network

Quantity	Type	Index, Subindex	Description
64	INT8	0xA000, 0x01…0x10	Signed 8-Bit object
64	UINT8	0xA040, 0x01…0x10	Unsigned 8-Bit object
64	INT16	0xA0C0, 0x01…0x10	Signed 16-Bit object
64	UINT16	0xA100, 0x01…0x10	Unsigned 16-Bit object
64	INT32	0xA1C0, 0x01…0x10	Signed 32-Bit object
64	UINT32	0xA200, 0x01…0x10	Unsigned 32-Bit object
32	INT64	0xA400, 0x01…0x10	Signed 64-Bit object
32	UINT64	0xA440, 0x01…0x10	Unsigned 64-Bit object

Table 7-122 Process Input Objects (Object Dictionary)

7.2.2 Process Outputs

Direct Output Variables Slave Network (Communication)

Quantity	Type	Address (BUS 2)	Description
64	SINT	AT %QB 2.0.0.0 - 2.0.15.0	Signed 8-Bit variable
64	USINT	AT %QB 2.1.0.0 - 2.1.15.0	Unsigned 8-Bit variable
64	INT	AT %QW 2.2.0.0 - 2.2.31.0	Signed 16-Bit variable
64	UINT	AT %QW 2.3.0.0 - 2.3.31.0	Unsigned 16-Bit variable
64	DINT	AT %QD2.4.0.0 - 2.4.61.0	Signed 32-Bit variable
64	UDINT	AT %QD2.5.0.0 - 2.5.61.0	Unsigned 32-Bit variable
32	LINT	AT %QL2.6.0.0 - 2.6.120.0	Signed 64-Bit variable
32	ULINT	AT %QL2.7.0.0 - 2.7.120.0	Unsigned 64-Bit variable

Table 7-123 Output Network Variables (IEC-61131 Program)

Process Output Objects Slave Network

Quantity	Type	Index, Subindex	Description
64	INT8	0xA480, 0x01-0x10	Signed 8-Bit object
64	UINT8	0xA4C0, 0x01-0x10	Unsigned 8-Bit object
64	INT16	0xA540, 0x01-0x10	Signed 16-Bit object
64	UINT16	0xA580, 0x01-0x10	Unsigned 16-Bit object
64	INT32	0xA640, 0x01-0x10	Signed 32-Bit object
64	UINT32	0xA680, 0x01-0x10	Unsigned 32-Bit object
32	INT64	0xA880, 0x01-0x10	Signed 64-Bit object
32	UINT64	0xA8C0, 0x01-0x10	Unsigned 64-Bit object

Table 7-124 Process Output Objects (Object Dictionary)

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	➔EPOS2 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	➔EPOS2 P error history for details
1908	System was reset by watchdog	Warning only*1)
1909	Interrupt Task error	Previous interrupt processing has not yet finished
1911	Execution error: data or program exception	Fatal application processing error
1913	Data History Buffer Overrun	Warning from CDA (sampling rate and/or number of variables should be reduced)
2001	RUN TIME ERROR: division by zero	
2002	RUN TIME ERROR: invalid array index	
2003	RUN TIME ERROR: invalid opcode	Unsupported command

Error Code	Description	Comment
2004	RUN TIME ERROR: opcode not supported	Unsupported command
2005	RUN TIME ERROR: invalid extension	Unsupported command
2006	RUN TIME ERROR: unknown command	Unsupported command
2008	Invalid bit reference	Runtime error
2009	Error restoring data	Runtime error
2010	Invalid array element size	Runtime error
2011	Invalid struct size	Runtime error
2012	RUN TIME ERROR: modulo zero, result undefined	

Remark

1) «EPOS Studio» also uses the watchdog to reset the node. Therefore, this warning may also be triggered as the EPOS Studio manipulates the EPOS2 P.

Table 8-125 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 (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 CiA 301, CiA 402, etc).	➔separate document «EPOS2 Firmware Specification»
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	

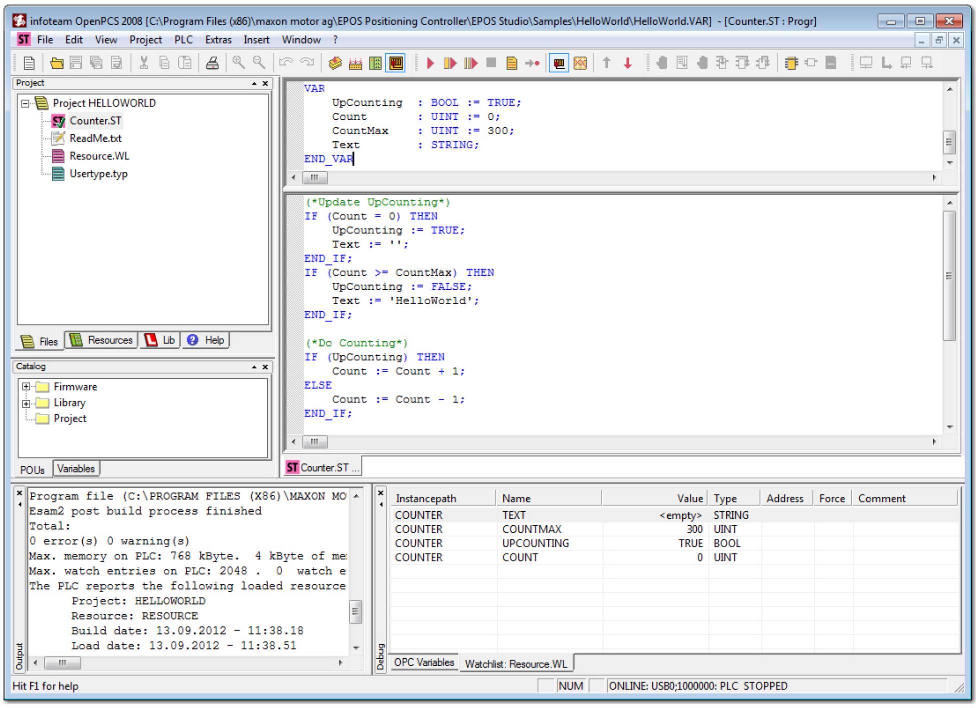
Table 8-126 Error Codes – Motion Control Function Blocks

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 EPOS2 P.	
Used Languages	Structured Text	
Task	Timer Task (10 ms)	
Files	Project file Main program Additional information	HelloWorld.VAR Counter.ST ReadMe.TXT

Table 9-127 «HelloWorld» in Brief


infoteam OpenPCS 2008 [C:\Program Files (x86)\maxon motor ag\EPOS Positioning Controller\EPOS Studio\Samples\HelloWorld\HelloWorld.VAR] - [Counter.ST : Progr]
File Edit View Project PLC Extras Insert Window ?
Project
Project HELLOWORLD
Counter.ST
ReadMe.txt
Resource.WL
Usertype.typ
Files Resources Lib Help
Catalog
Firmware
Library
Project
POUs Variables
Counter.ST ...
Output
Debug
OPC Variables Watchlist: Resource.WL
Hit F1 for help
NUM
ONLINE: USB0;1000000: PLC STOPPED

```
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 := 'HelloWorld';
END_IF;

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

```
Program file (C:\PROGRAM FILES (X86)\MAXON MO
Esam2 post build process finished
Total:
0 error(s) 0 warning(s)
Max. memory on PLC: 768 kByte.  4 kByte of me
Max. watch entries on PLC: 2048 .  0  watch e
The PLC reports the following loaded resource
      Project: HELLOWORLD
      Resource: RESOURCE
      Build date: 13.09.2012 - 11:38.18
      Load date: 13.09.2012 - 11:38.51
```

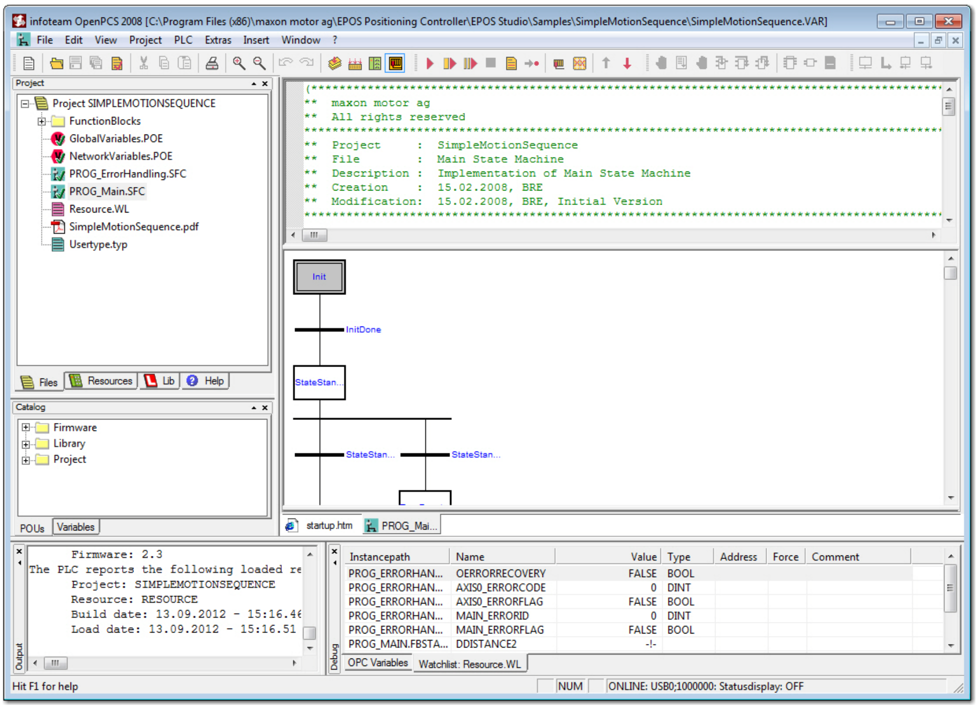
Instancepath	Name	Value	Type	Address	Force	Comment
COUNTER	TEXT	<empty>	STRING			
COUNTER	COUNTMAX	300	UINT			
COUNTER	UPCOUNTING	TRUE	BOOL			
COUNTER	COUNT	0	UINT			

Table 9-128 «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 9-129 «SimpleMotionSequence» in Brief



```
(*****************************************************************************
**  maxon motor ag
**  All rights reserved
******************************************************************************
**  Project     :  SimpleMotionSequence
**  File        :  Main State Machine
**  Description :  Implementation of Main State Machine
**  Creation    :  15.02.2008, BRE
**  Modification:  15.02.2008, BRE, Initial Version
******************************************************************************
```

```
      Firmware: 2.3
The PLC reports the following loaded re
      Project: SIMPLEMOTIONSEQUENCE
      Resource: RESOURCE
      Build date: 13.09.2012 - 15:16.46
      Load date: 13.09.2012 - 15:16.51
```

Instancepath	Name	Value	Type	Address	Force	Comment
PROG_ERRORHAN...	OERRORRECOVERY	FALSE	BOOL			
PROG_ERRORHAN...	AXIS0_ERRORCODE	0	DINT			
PROG_ERRORHAN...	AXIS0_ERRORFLAG	FALSE	BOOL			
PROG_ERRORHAN...	MAIN_ERRORID	0	DINT			
PROG_ERRORHAN...	MAIN_ERRORFLAG	FALSE	BOOL			
PROG_MAIN.FBSTA...	DDISTANCE2	-!-				

Table 9-130 «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 EPOS2 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 EPOS2 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 9-131 Best Practice Program Examples

9.4 Application Program Examples

The example collection shows complete applications of EPOS2 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».
«Process Input Output»	The example demonstrates how to implement a supervisory control application. For details ➔separate document «ProcessInputOutputProject.pdf».

Table 9-132 Application Program Examples

LIST OF FIGURES

••page intentionally left blank••

LIST OF TABLES

••page intentionally left blank••

INDEX

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

Phone +41 41 666 15 00
Fax +41 41 666 16 50

www.maxonmotor.com