

AC500 PLC

AC500 V3 Motion Controller Guide

Motion Controller with AC500 V3

Documentation

NOTICE

This document contains information about one or more ABB products and may include a description of or a reference to one or more standards that may be generally relevant to the ABB products. The presence of any such description of a standard or reference to a standard is not a representation that all of the ABB products referenced in this document support all of the features of the described or referenced standard. In order to determine the specific features supported by a particular ABB product, the reader should consult the product specifications for the particular ABB product.

ABB may have one or more patents or pending patent applications protecting the intellectual property in the ABB products described in this document.

The information in this document is subject to change without notice and should not be construed as a commitment by ABB. ABB assumes no responsibility for any errors that may appear in this document.

Products described or referenced in this document are designed to be connected and to communicate information and data through network interfaces, which should be connected to a secure network. It is the sole responsibility of the system/product owner to provide and continuously ensure a secure connection between the product and the system network and/or any other networks that may be connected.

The system/product owners must establish and maintain appropriate measures, including, but not limited to, the installation of firewalls, application of authentication measures, encryption of data, installation of antivirus programs, and so on, to protect these products, the network, its system, and interfaces against security breaches, unauthorized access, interference, intrusion, leakage, and/or theft of data or information.

ABB performs functionality testing on the products and updates that we release. However, system/product owners are ultimately responsible for ensuring that any product updates or other major system updates (to include but not limited to code changes, configuration file changes, third-party software updates or patches, hardware change out, and so on) are compatible with the security measures implemented. The system/ product owners must verify that the system and associated products function as expected in the environment in which they are deployed.

In no event shall ABB be liable for direct, indirect, special, incidental or consequential damages of any nature or kind arising from the use of this document, nor shall ABB be liable for incidental or consequential damages arising from use of any software or hardware described in this document.

This document and parts thereof must not be reproduced or copied without written permission from ABB, and the contents thereof must not be imparted to a third party nor used for any unauthorized purpose.

The software or hardware described in this document is furnished under a license and may be used, copied, or disclosed only in accordance with the terms of such license. This product meets the requirements specified in EMC Directive 2014/30/EU and in Low Voltage Directive 2014/35/EU.

A. For customers domiciled outside Germany /

Für Kunden mit Sitz außerhalb Deutschlands

„Warranty, Liability:

The user shall be solely responsible for the use of the products described within this file. ABB shall be under no warranty whatsoever. ABB's liability in connection with application of the products or examples provided or the files included within these products, irrespective of the legal ground, shall be excluded. The exclusion of liability shall not apply in the case of intention or gross negligence. The present declaration shall be governed by and construed in accordance with the laws of Switzerland under exclusion of its conflict of laws rules and of the Vienna Convention on the International Sale of Goods (CISG)."

„Gewährleistung und Haftung:

Der Nutzer ist allein für die Verwendung des in diesem Dokument beschriebenen Produkte und beschriebenen Anwendungsbeispiele verantwortlich.

ABB unterliegt keiner Gewährleistung. Die Haftung von ABB im Zusammenhang mit diesem Anwendungsbeispiel oder den in dieser Datei enthaltenen Dateien - gleich aus welchem Rechtsgrund - ist ausgeschlossen. Dieser Ausschluß gilt nicht im Falle von Vorsatz oder grober Fahrlässigkeit. Diese Erklärung unterliegt Schweizer Recht unter Ausschluß der Verweisungsnormen und des UN-Kaufrechts (CISG)."

B. Nur für Kunden mit Sitz in Deutschland

„Gewährleistung und Haftung:

Die in diesem Dokument beschriebenen Anwendungsbeispiele oder enthaltenen Dateien beschreiben eine mögliche Anwendung der AC500 bzw. zeigen eine mögliche Einsatzart. Sie stellen nur Beispiele für Programmierungen dar, sind aber keine fertigen Lösungen. Eine Gewähr kann nicht übernommen werden.

Der Nutzer ist für die ordnungsgemäße, insbesondere vollständige und fehlerfreie Programmierung der Steuerungen selbst verantwortlich. Im Falle der teilweisen oder ganzen Übernahme der Programmierbeispiele können gegen ABB keine Ansprüche geltend gemacht werden.

Die Haftung von ABB, gleich aus welchem Rechtsgrund, im Zusammenhang mit den Anwendungsbeispielen oder den in dieser Datei enthaltenen Beschreibung wird ausgeschlossen. Der Haftungsausschluß gilt jedoch nicht in Fällen des Vorsatzes, der groben Fahrlässigkeit, bei Ansprüchen nach dem Produkthaftungsgesetz, im Falle der Verletzung des Lebens, des Körpers oder der Gesundheit oder bei schuldhafter Verletzung einer wesentlichen Vertragspflicht. Im Falle der Verletzung einer wesentlichen Vertragspflicht ist die Haftung jedoch auf den vertragstypischen, vorhersehbaren Schaden begrenzt, soweit nicht zugleich ein anderer der in Satz 2 dieses Unterabsatzes erwähnten Fälle gegeben ist. Eine Änderung der Beweislast zum Nachteil des Nutzers ist hiermit nicht verbunden.

Es gilt materielles deutsches Recht unter Ausschluß des UN-Kaufrechts."

TRADEMARKS

All rights to copyrights, registered trademarks, and trademarks reside with their respective owners.

Copyright © 2022 ABB.

All rights reserved.

Release: December 2022

Document number: 3ADR011116

1 Contents

1 INTRODUCTION11

1.1 Scope of the document11

1.2 Safety Instructions and Preconditions11

2 AC500 PRODUCT OVERVIEW 13

2.1 AC500 PLC overview 13

2.2 AC500 / S500 hardware overview 14

2.3 Selecting an AC500 “V3” CPU as Motion Controller 14

2.3.1 Identifying AC500 “V3” CPU..... 14

2.3.2 Understanding the ABB products type codes and labels: 15

2.3.3 Understanding the Contents of ABB Motion Controller Kits:..... 16

2.3.4 The CPUs main technical data and limits for motion control selection..... 17

2.4 Mechanical installation..... 18

2.4.1 Mounting and demounting 19

2.5 Electrical connection 33

2.5.1 Power supply for processor modules 33

2.6 CPU function keys Display and LED display 34

2.6.1 Description of the function keys34

2.6.2 Description of Display35

2.6.3 Other common display codes36

2.6.4 Description of LED36

2.7 Accessories (TA521 - Lithium battery) 36

2.8 Introduction to ABB PLC Licenses 37

3 AUTOMATION BUILDER OVERVIEW 39

3.1 Software installation 39

3.1.1 Preconditions39

3.1.2 Online Installation42

3.1.3 Offline Installation.....47

3.1.4 Installing additional tools55

3.2 Software user licensing of Automation Builder 56

3.2.1 Online Activation58

3.2.2 Offline Activation.....59

3.3 Using Servo Drives with AC500 PLC 62

3.3.1 Setting up ABB Servo Drives for use with EtherCAT Master62

3.3.2 Exporting the xml file from the drive63

3.3.3 Adding ABB and 3rd party devices to the Device repository63

4 INTRODUCTION TO THE PROJECT 66

4.1 Project types guidance..... 66

4.1.1	Different project types	66
4.1.2	Understanding when to use the different project types	67
4.2	Selecting hardware used in the project	67
4.2.1	Select PLC Type.....	67
4.2.2	Saving the project	68
4.2.3	Navigating the project.....	69
4.3	Important CPU parameters	69
4.3.1	Checking program size and number of configured axis.....	70
4.4	Changing CPU type	71
4.5	I/O in AC500 and S500 IO System	73
4.5.1	Configuring local ABB I/O module (S500).....	73
4.5.2	Configuring to ABB Remote IO.....	76
4.5.3	Configuring to 3rd Party Remote IO.....	78
4.6	Fieldbus protocol types	79
4.6.1	Communication using Onboard Ethernet Ports	79
4.6.2	Communication via a coupler Communications module.....	80
4.7	Programming and compiling AC500 code	82
4.8	Library Manager Introduction	85
4.8.1	Add or Search function.....	85
4.8.2	Placeholders and handling different library versions.....	86
4.8.3	Library Repository	87
4.8.4	View embedded documentation of all libraries	87
4.9	Task configuration.....	88
4.9.1	Understanding Task Configuration.....	88
4.9.2	Task types and task monitor.....	89
4.10	Real time clock and battery	89
4.11	Integrated project visualization	89
4.11.1	Add the Visualization	89
4.11.2	Set-up the Visualization Manager.....	91
4.11.3	Enable web visualization	91
5	AC500 COMMUNICATION PROTOCOLS	95
5.1	Supported Protocols Overview	95
5.2	EtherCAT.....	95
5.2.1	Configuring the CM579-ETHCAT EtherCAT master in the project.....	95
5.2.2	CM579-ETHCAT	95
5.2.3	EtherCAT Master Settings	96
5.2.4	EtherCAT Slave Settings.....	100
5.2.5	Setting up the PLC and ABB Servo EtherCAT Slave for EoE Comms	105
5.2.6	How to add a Serial Protocol.....	109

5.2.7	Modbus RTU Server (Slave)	109
5.2.8	Modbus RTU Client (Master)	110
5.2.9	HMI Modbus RTU communication	110
5.3	Modbus TCP/IP.....	111
5.3.1	Modbus TCP/IP Server.....	112
5.3.2	Modbus TCP/IP Client	113
5.3.3	HMI's and Modbus TCP/IP communication	114
5.4	OPC UA.....	115
6	GETTING ONLINE AND MANAGING THE PLC.....	118
6.1	Getting online to the PLC	118
6.1.1	Set-up communication parameters in windows.....	118
6.1.2	Configuration of the PLC IP settings	119
6.1.3	Set-up the communication gateway	119
6.1.4	Check communication settings	121
6.1.5	Change PLC IP address.....	121
6.2	Login to the CPU and download the program.....	123
6.3	Firmware update	124
6.3.1	Behaviour of LEDs during firmware update	125
6.4	Run time license for PLC for Motion Control.....	126
6.4.1	What is Run time licensing.....	126
6.4.2	Activating PLC license with internet connection	126
6.4.3	Downloading and activating PLC license without internet connection.....	127
6.4.4	Downloading and activating PLC license via memory card	129
6.4.5	Activating a demo license	130
6.4.6	Returning a license from a PLC	130
7	GENERAL PLC PROGRAM BASICS	133
7.1	Programming languages and editors	133
7.2	Variable classifications	133
7.2.1	Local Variables - VAR.....	133
7.2.2	Input Variables - VAR_INPUT.....	133
7.2.3	Output Variables - VAR_OUTPUT	133
7.2.4	Input/Output Variable (VAR_IN_OUT)	134
7.2.5	Global Variables - VAR_GLOBAL	134
7.2.6	Temporary Variable - VAR_TEMP	134
7.2.7	Static Variables - VAR_STAT	135
7.2.8	Constant Variables - 'CONSTANT'	135
7.2.9	Persistent Variable - PERSISTENT.....	135
7.2.10	Retain Variable - RETAIN.....	136
7.2.11	Handling of remanent variables for AC500 V3 products	136

7.3	Data types	136
7.3.1	BOOL.....	136
7.3.2	INTEGER	136
7.3.3	REAL / LREAL.....	137
7.3.4	STRING.....	137
7.3.5	TIME	137
7.3.6	LTIME	138
7.3.7	Date and Time	138
7.3.8	BIT	139
7.3.9	Pointers	139
7.3.10	ARRAY	140
7.3.11	SUM := diResult;Structure (STRUCT).....	146
7.3.12	Enumerations (ENUM)	149
7.4	ST Statements	153
7.4.1	IF	153
7.4.2	FOR.....	154
7.4.3	CASE.....	154
7.4.4	WHILE	155
7.5	REPEAT.....	156
7.5.1	RETURN	156
7.5.2	JMP	156
7.5.3	EXIT	157
7.5.4	CONTINUE.....	157
7.5.5	Function Block Call	157
8	MOTION SOLUTION PROJECT	158
8.1	Introduction	158
8.1.1	Understanding the Motion Solution Project.....	158
8.1.2	Understanding the Motion Solution Wizard	158
8.1.3	Understanding the Axis Objects	158
8.2	Installing the latest Motion Control Wizard and Libraries	158
8.3	Creating new Motion Solution project.....	158
8.3.1	Creating new project	158
8.3.2	Add PLC types	159
8.3.3	Add PTO axis	160
8.3.4	Add EtherCAT axis	167
8.3.5	Adding encoder axis.....	176
8.3.6	Adding virtual axis	178
8.4	Motion Axis generation	179
8.4.1	PTO axis.....	180

8.4.2	EtherCAT motion axis	183
8.4.3	Axis program generated (Hidden by default)	186
8.5	Writing Application program	190
9	CAM EDITOR	192
9.1	Definition of a Cam	192
9.2	Structure of the Cam Editor	192
9.2.1	Tab 'Cam'	192
9.2.2	Tab 'Cam table'	193
9.2.3	Tab 'Tappets'	193
9.2.4	Tab 'Tappet table'	195
9.2.5	Dialog 'Properties - 'Cam'	195
9.3	Creating Cams	196
9.3.1	Adding a cam to the device tree	196
9.3.2	Setting the properties of the cam	197
9.3.3	Changing the Cam Path	197
9.3.4	Defining Switch Points	198
9.4	Cam generated code	199
9.5	Importing a Cam from 3rd party Codesys controller	200
9.5.1	Exporting the Cam for the the 3 rd party PLC	200
9.5.2	Importing the Cam data into Automation Builder	202
9.6	Application program using generated Cam	204
10	ABB PLCOPEN MOTION CONTROL LIBRARY	206
10.1	Motion Control library: System Technology	206
10.1.1	Preconditions for the use of the motion control libraries	206
10.1.2	Overview and Basics	207
10.1.3	PLCopen Introduction and Basics	215
10.1.4	PLC-based motion control	235
10.1.5	Load Control/Torque Control: Fluid Power Extension according PLCopen	271
10.2	PLCopen based Motion Control Libraries (Function Block descriptions)	277
10.2.1	MotionControl (Library)	277
10.2.2	MotionControlLoad (Library)	436
10.2.3	MotionControlEco (Library)	460
10.2.4	Ecat_CiA402 (Library)	468
10.2.5	MathFunctions (Library)	485
11	DIAGNOSIS	488
11.1	Online diagnosis using Automation Builder	488
11.2	Diagnosis in PLC program	489
11.3	EtherCAT Diagnosis	489
11.3.1	Application scenarios of EtherCAT diagnostics	489

- 11.3.2 Operational..... 490
- 11.3.3 Diagnostic with Automation Builder491
- 11.3.4 Process guideline for typical faults and errors during commissioning 495
- 11.4 Diagnostic with IEC programming498**
- 11.4.1 Topology error 499
- 11.4.2 Communication error501
- 11.5 Data recording with trace502**
- REVISION HISTORY506**

1 INTRODUCTION



Note: Please refer to ABB library for the latest version of Motion Controller Guide [3ADR011116](#)

1.1 Scope of the document

This document contains the most important parts out of the complete AC500 documentation in a compact form to enable an efficient start when using the AC500 V3 PLC as a motion controller.

It contains a compact overview of the AC500 platform including hardware and engineering of the options with Automation Builder as central engineering tool and then mainly focusses on the parts needed for typical real time motion control applications using EtherCAT or PTO type motion drives.

The “Motion Control” package needs to be installed as a separate option from Automation Builder Installation Manager. This will install the below features:

- Motion Solution Wizard
- Cam Editor
- Motion Control PLCopen library, where above engineering tools are fully based on.



Note: This software manual carries the information which are needed for an engineer to start a motion application quickly. For the latest and more detailed information on AC500 hardware and software, please refer the latest Automation Builder integrated help file (or its [pdf versions in ABB Library](#)).

The Motion Solution wizard is an integrated tool within Automation Builder. It improves the setup, overview and time needed for axis configuration significantly, so that the users can focus more on the motion tasks. This is then possible by easy configuration of different application parameters and adding further PLCopen function blocks of Motion Control library (PS5611-MC) - based on the application needs

The Cam editor tool helps to create Cam table(s), which can be easily created using the graphical window of Cam editor. The Cam path(s) can be created offline in an integrated graphical window or a linked Cam table to quickly create the Cam points as per the application requirements.

Functional description of the PLCopen principles used, the Motion Control libraries and details of the contained blocks are available in this document.

Note! *The material in this application might need to be adapted according to actual equipment and function before it is used. Testing of the equipment must always be performed by the responsible start-up person according to current legislation before the equipment is placed in service. ABB does not take any responsibility for possible damage caused by using the material in this application (shown examples, data, project tools, etc.).*

1.2 Safety Instructions and Preconditions

The user must read the following instructions and documents before using the libraries:

All pertinent state, regional, and local safety regulations must be observed when installing and using this product. When functions or devices are used for applications with technical safety requirements, the relevant instructions must be followed.

Read the complete safety instructions of the user's manuals for the devices you are using before installation and commissioning.

Read all safety instructions of the AC500 PLC. See System description AC500 in the online help in Automation Builder

Read the user Information of the devices and functions you are using, see online help in Automation Builder.

The Motion Control package has been released for the software and firmware versions listed in the Readme file of the package only.

In no event will ABB or its representatives be liable for loss of data, profits, revenue or consequential, incidental, or other damage that may result from the use of other versions of product, software or firmware versions. The error-free operation of the Motion Control Package with other devices, software or firmware versions should be possible but cannot be guaranteed and may need adaptations e.g. of example programs.

The user must follow all applicable safety instructions and the guidelines mentioned in the user documents of the ABB products.

Read the complete safety instructions for the AC500 before installation and commissioning.



CAUTION!

Generally, the user in all applications is fully and alone responsible for checking all functions carefully, especially for safe and reliable operation.

2 AC500 PRODUCT OVERVIEW

ABB has a wide PLC Automation portfolio, with a range of scalable and robust PLCs and HMI control panels, with one joint engineering suite to provide solutions for small, medium and high-end applications.

Features and options allow tailoring for various industries and applications including water, building infrastructure, data centers, renewable energy, machinery automation and motion control, material handling, marine to name a few.

2.1 AC500 PLC overview

Engineering suite: Automation Builder

Automation Builder is the integrated software suite for machine builders and system integrators requiring state-of-the-art productive machine and system automation. This combines the tools required for configuring, programming, debugging and maintaining automation projects efficiently from one common intuitive interface.

Programmable Logic Controllers PLCs:

The platform offers interoperability, and compatibility in hardware and software from compact AC500-eCo PLCs up to high end AC500 PLC's.

The newest range of CPU is called V3 as it bases on the V3 CODESYS platform. This range offers different performance levels together with its wide S500 and S500 eCo IO assortment, while its XC variants cover extreme environments.

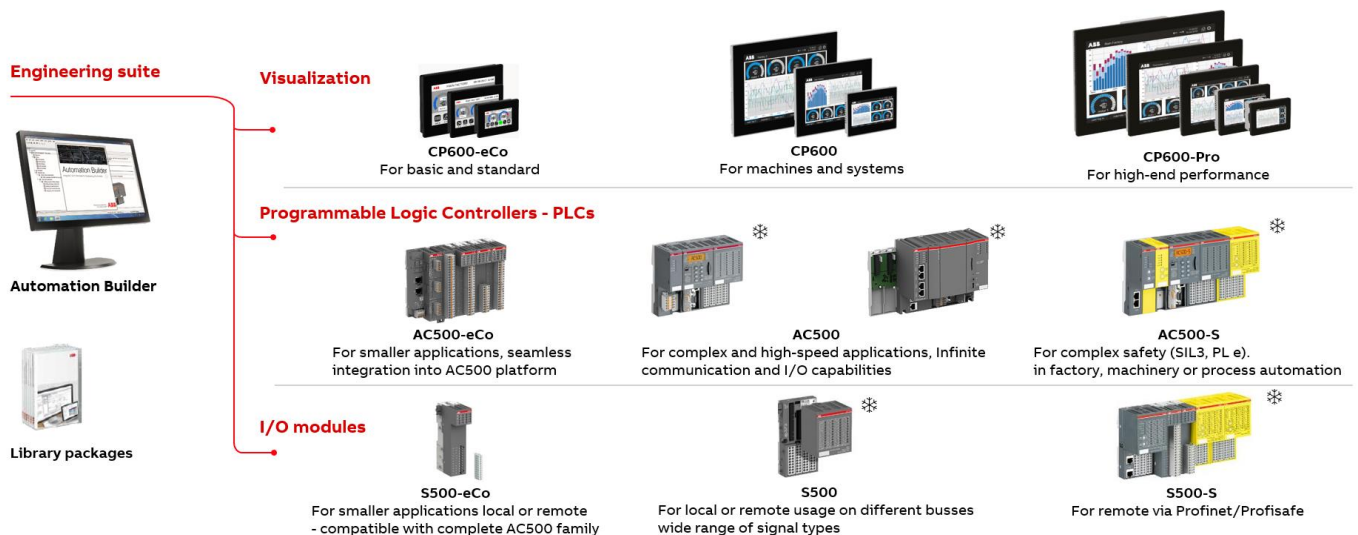
The platform offers interoperability, and compatibility in hardware and software from compact PLCs up to high end and safety PLCs.

AC500 Motion Control Kits are packages of hardware which make the AC500 the easy to order, perfect motion controller.

Control panels, visualization:

The CP600-eCo, CP600 and CP600-Pro control panels in combination with the PB610 Panel Builder offer a wide range of features and functionalities for inter-operability.

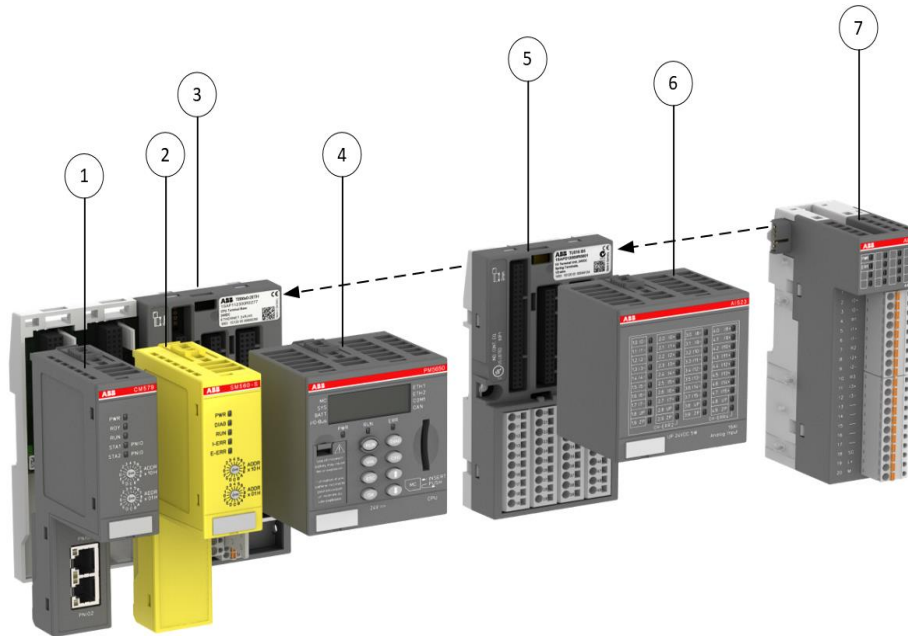
ABB control panels are distinguished by their robustness and easy usability, providing all the relevant information from production plants and machines at one single touch.



2.2 AC500 / S500 hardware overview

The AC500 platform provides flexibility:

- CM Communication-Modules (1) can be added on the left side to AC500 CPU (4) via the Terminal Base (3; a TB specific number of slots).
- S500 I/O- modules (6) or S500eco I/O modules (7) can be added on the right side to AC500 (4) via TU Terminal Units (5) (or to an AC500eCo CPU); max. 10 modules .



- (1) Communication modules CM5xx
- (2) Safety PLC CPU SM5xx
- (3) Terminal base for CPUs and CM5xx
V3: TB5xxx
- (4) CPU: AC500 central processing unit
V3: PM5xxx

- (5) S500 Terminal unit: TU5xx
- (6) S500 I/O-module: XYxxx
- (7) S500 eCo I/O-module:
XYxxx

Features of AC500 Platform

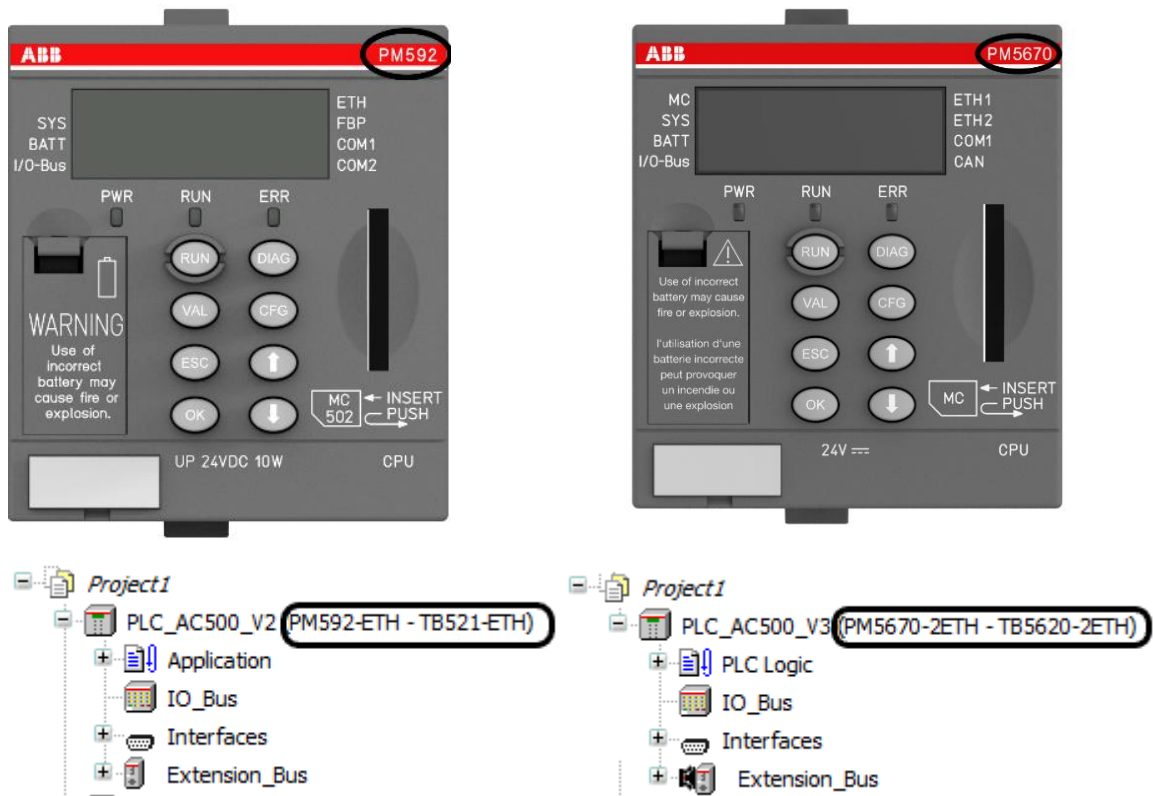
- Scalable and expandable system
- Different performance classes of processor modules (CPUs) available
- Several integrated onboard protocols which can run also in parallel (e.g. OPC + Modbus + UDP, ...), some of them licensed.
- Several *additional*/field busses available via the CM communication modules (AC500 only)
- Parallel connection to several field busses which can be combined arbitrarily

2.3 Selecting an AC500 “V3” CPU as Motion Controller

2.3.1 Identifying AC500 “V3” CPU

In principle there are two CPU ranges the older V2, and the newer V3 version based on the respective CODESYS Vx platform.

The advanced Motion Control solution with Motion wizard and Cam editor is only supported in AC500 V3 PLC. You can easily find out which type you are looking at either by looking at your used modules or your Automation Builder configuration.



CPU type written on the module / Configurator	Example type	CPU range
CPU type PM + 3 digits	PM592	V2
CPU type PM + 4 digits	PM5670	V3

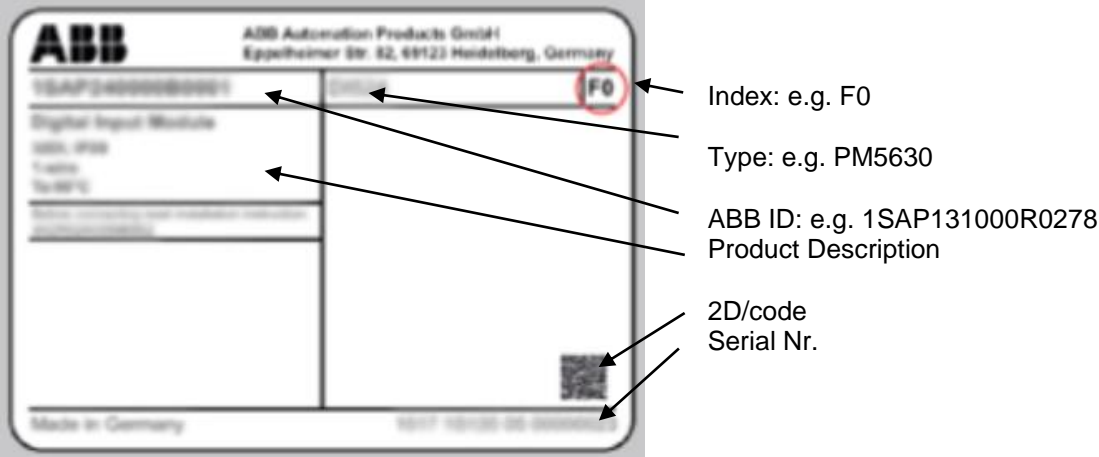
Relevant manual can be found at Automation Builder help, under chapter PLC Automation with V2 CPUs & PLC Automation with V3 CPUs respectively.

2.3.2 Understanding the ABB products type codes and labels:

Each ABB Device has a product type, a version and an ABB ID.

- The **“Product Type”** is shown normally on the front side of the product in the form PMXXYY-ZETH for example **“PM5650-2ETH”**
 - XX shows the CPU range, for example if PM56YY = AC500 or if PM50YY = AC500eco,
 - YY shows the processor and memory variant, see later chapters for more information on this.
 - ZETH shows the number of supported onboard Ethernet ports for example **“PMXXYY”-“2ETH”** shows the CPU supports two onboard Ethernet ports.
- The version index shows which hardware version it is. Different versions of a product over its life-time might be indexed only on the label next to the ABB ID on the device itself

by a following capital letter/number (e.g. “PM5630-2ETH “F0”)



- The **ABB ID** or ‘part- number’ is used in database searches and ordering and is shown in the form of a code like : e.g. 1SAP131000R0278

Each of above identifies the product uniquely.

The labels on the products and packages also contain an EAN identification number and a serial number with their 2D/3D codes.

2.3.3 Understanding the Contents of ABB Motion Controller Kits:

For simplified ordering of AC500 parts needed for motion control applications, there are Kits available, which bundle the minimum parts needed for a motion controller: e.g. EtherCAT communication module, Terminal base, CPU and Motion License.

AC500 Motion Controller Kits

Product	Order code	Contains
PM5630-MC-KIT	1SAP131000R0379	PM5630-2ETH 80MB CPU CM579-ETHCAT communication module TB5610-ETH terminal base PS5611-MC PLCopen® motion control runtime license
PM5650-MC-KIT	1SAP141000R0379	PM5650-2ETH 80MB CPU CM579-ETHCAT communication module TB5610-ETH terminal base PS5611-MC PLCopen® motion control runtime license
PM5670-MC-KIT	1SAP151000R0379	PM5670-2ETH 160MB CPU CM579-ETHCAT communication module TB5610-ETH terminal base PS5611-MC PLCopen® motion control runtime license

...

.

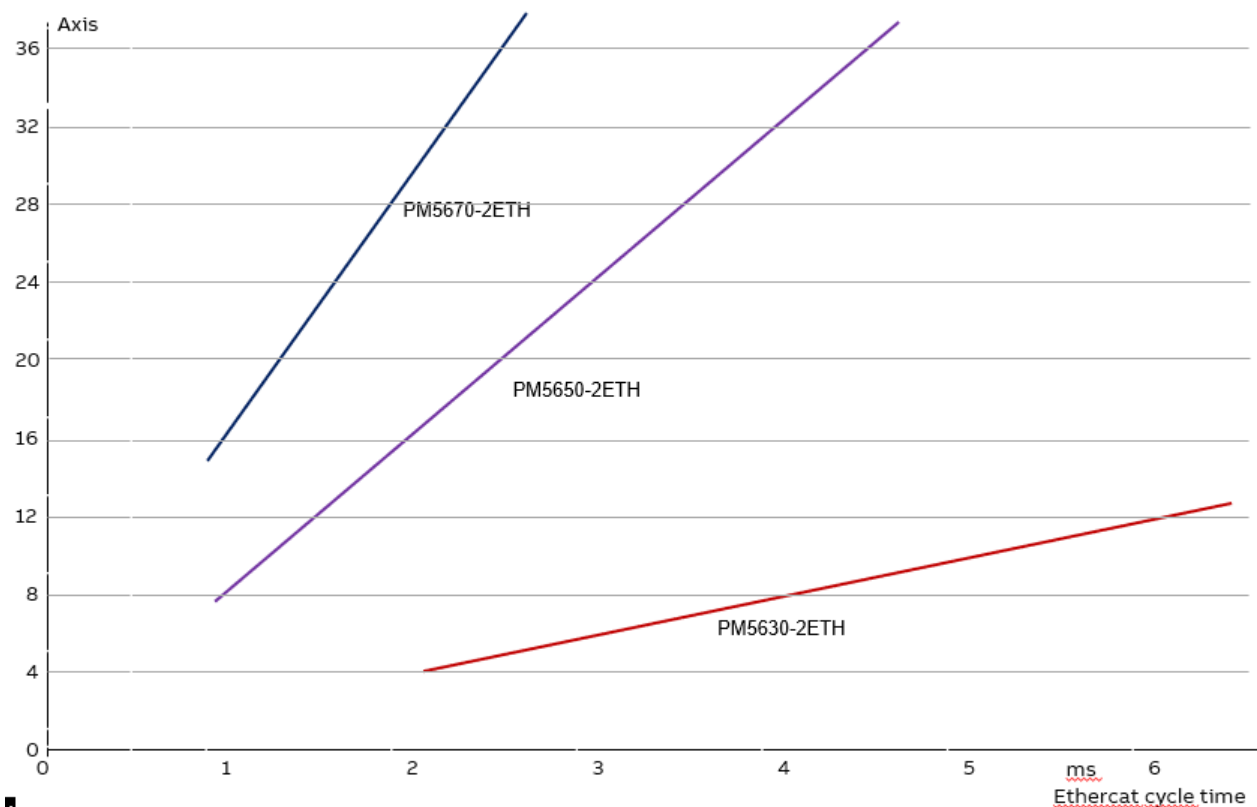
2.3.4 The CPUs main technical data and limits for motion control selection

1. Minimum EtherCAT cycle time configurable for each PLC type.

PLC Type	PM5032	PM5052	PM5072	PM5630	PM5650	PM5670
Min. EtherCAT master cycle time configurable in ms				2	1	0.5
Min normal Task cycle usable in ms	5	2	1	1	1	0.5
PTO available (up to 4x100kHz or 2x200kHz)	yes					
ETH Ports	1	1	2	2	2	2
Option Slots	2	3	3			

2. The maximum configurable number of synchronized axis in each PLC type, is depending on the set Ethercat cycle time of the master and is therefore scaling as in below table and graph.

PLC Type		PM5630	PM5650	PM5670
Ethercat axis configuration limits and max. performance indication Examples:				
Number of synchr. axis in 1 ms		-	8	16
Number of synchr. axis in 2 ms		4	16	32
Number of synchr. axis in 4 ms		8	32	64



This graph shows the scaling of the max. configurable EtherCAT axis number in graphical form:

2.4 Mechanical installation

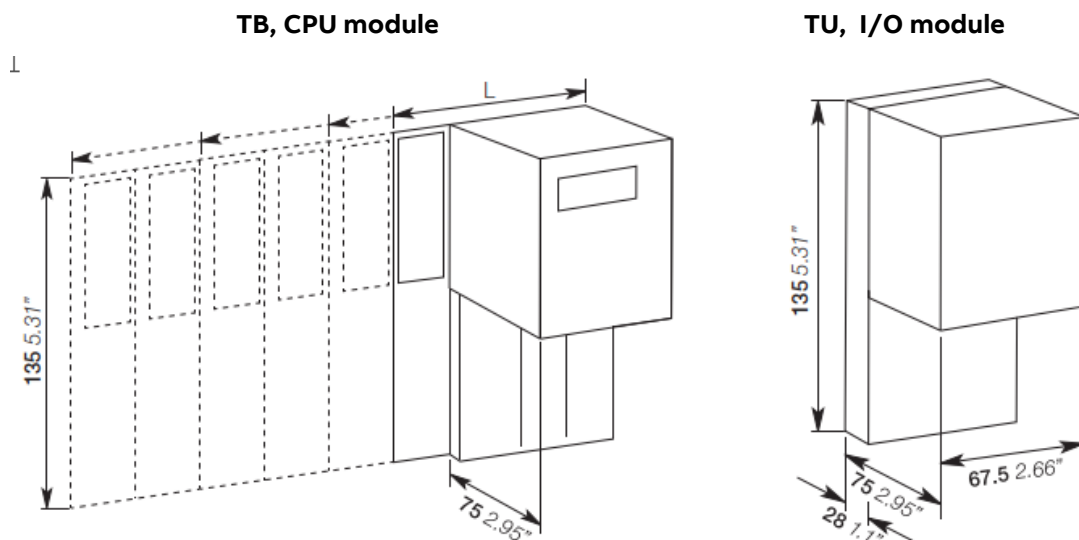


Note: For detailed updated information such as technical data of your mounted devices of AC500 product family - refer to the hardware device description of the appropriate device from the latest Automation Builder help file.

AC500 V3 and S500 IO main dimensions

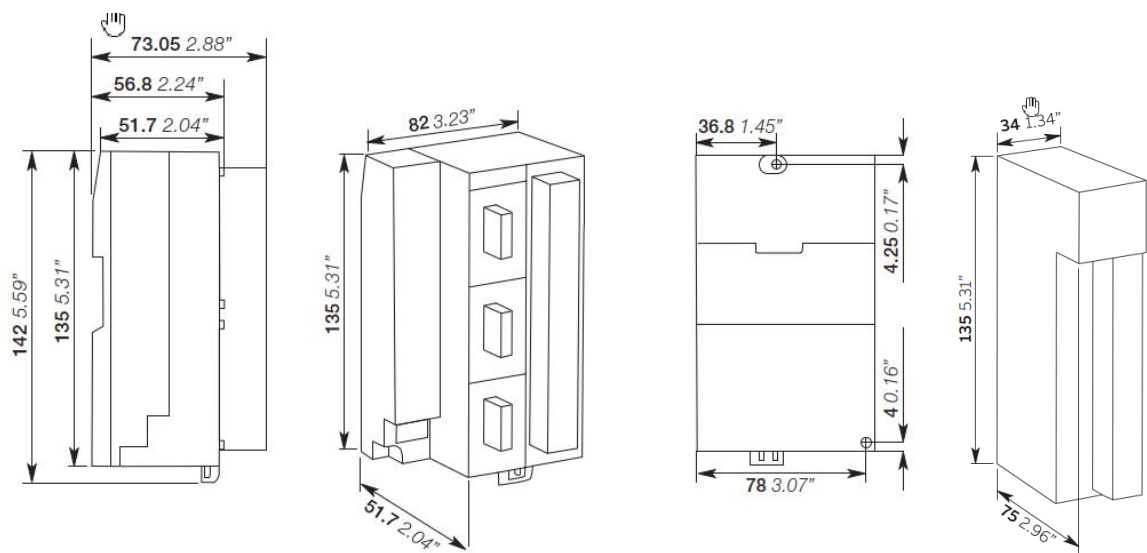
A terminal base **TB** is needed for mounting and connection of the AC500 V3 CPUs and communication modules. A terminal unit **TU** is needed for mounting and connection of the S500 I/O and communication interface modules.

They typically determine the cabinet footprint according below table and picture.





Type	Nr. communication modules	Length	L mm / inches
TB5600-2ETH	0		67.5 / 2.66
TB5610-2ETH	1		95.5 / 3.76
TB5620-2ETH	2		123.5 / 4.86
TB5640-2ETH	4		179.5 / 7.07
TB5660-2ETH	6		235.5 / 10.5

eCo V3 and S500 eCo main dimensions



2.4.1 Mounting and demounting

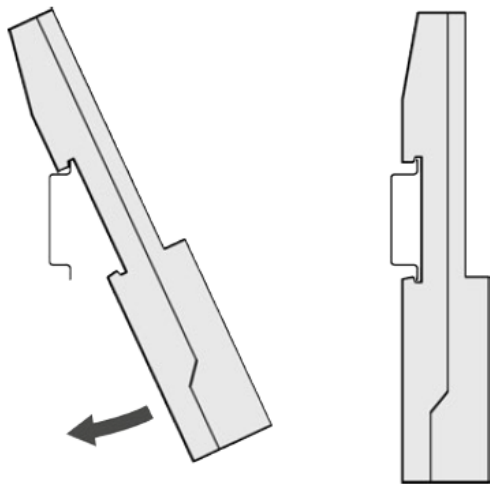
The control system is designed to be mounted to a well-grounded mounting surface such as a metal panel. Additional grounding connections from the mounting tabs or DIN rail (if used), are not required unless the mounting surface cannot be grounded. Mounting tabs of AC500 have a max. diameter suited for 4 mm screws.

	Note: During panel or DIN rail mounting of all devices, be sure that all debris (metal chips, wire strands, etc.) is kept from falling into the controller. Debris that falls into the controller could cause damage while the controller is energized.
	Note: All devices are grounded through the DIN rail to chassis ground. Use zinc plated yellow chromate steel DIN rail to assure proper grounding. The use of other DIN rail materials (e.g. aluminum, plastic, etc.) that can corrode, oxidize, or are poor conductors, can result in improper or intermittent grounding.

2.4.1.1 Terminal bases / unit mounting and demounting on DIN rail

Mount DIN rail 7.5 mm or 15 mm.

Mount the Terminal Base/Function Module Terminal Base:

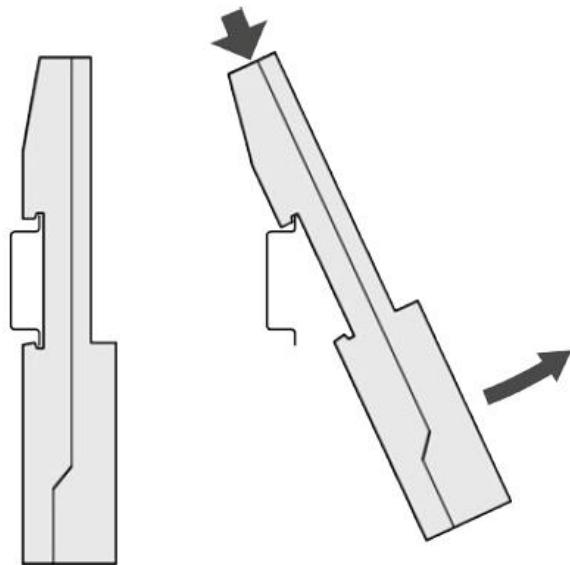


The Terminal Base is put on the DIN rail above and then snapped-in below.

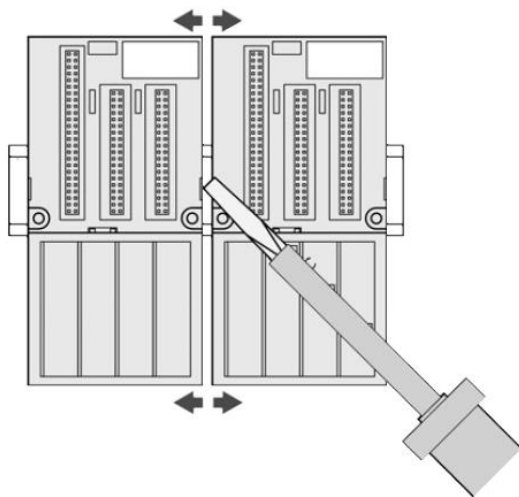


Note: When attaching the devices, make sure the bus connectors are securely locked together to ensure proper electrical connection. Max. 10 Terminal Units can be attached.

The demounting is carried out in a reversed order.



For separating terminal unit, a screwdriver is inserted in the indicated place to separate the Terminal Units.



2.4.1.2 Terminal bases / Terminal unit mounting with screws

If the Terminal Base should be mounted with screws (max. M4), Wall Mounting Accessories TA526 must be inserted at the rear side first. These plastic parts prevent bending of the Terminal Base while screwing on. TB560x and TB561x need one TA526, TB562x, TB564x and TB566x need two TA526.

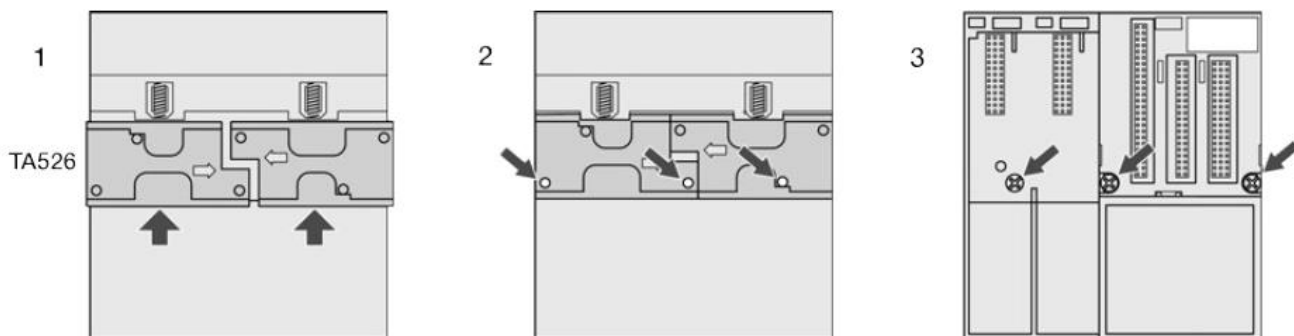


Figure - Terminal base, fastening with screws

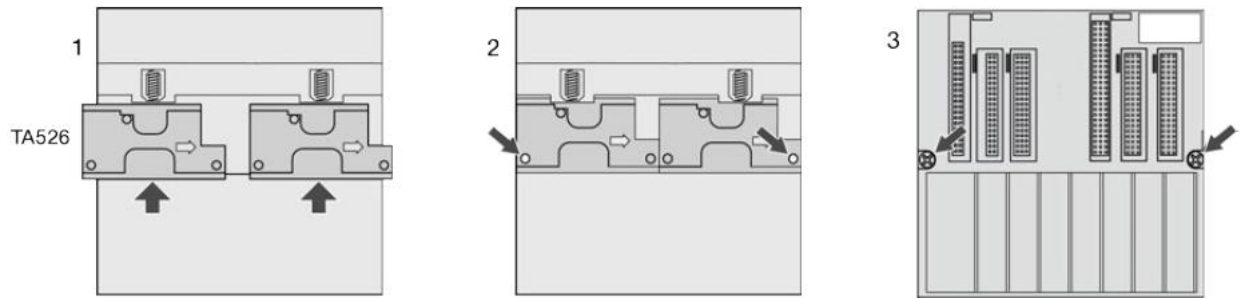


Figure - Function module terminal base, fastening with screws

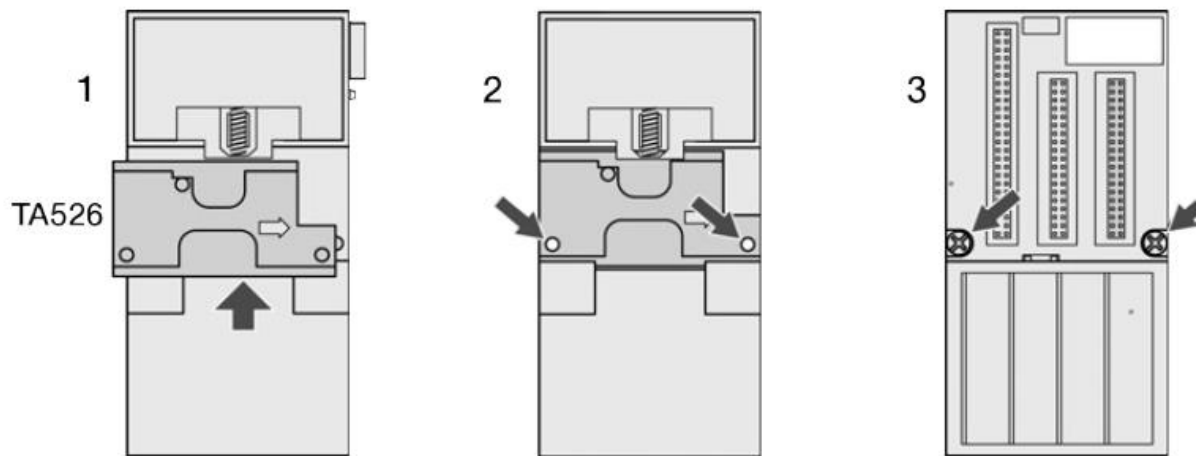
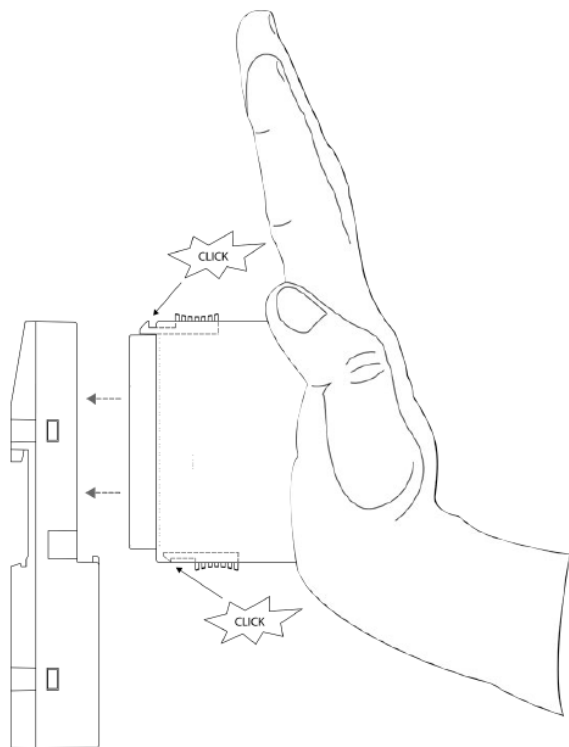


Figure - Terminal unit, fastening with screws

	<p>Note: By wall mounting, the Terminal Base is earthed through the screws. It is necessary that</p> <ul style="list-style-type: none"> The screws have a conductive surface (e.g. steel zinc-plated or brass nickel-plated). The mounting plate is earthed. The screws have a good electrical contact to the mounting plate.
--	---

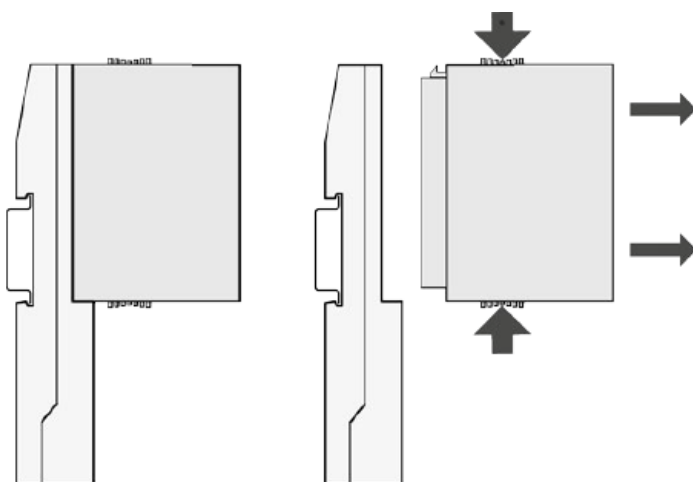
2.4.1.3 Mounting and demounting V3 processor (CPU) and S500 IO modules

After mounting the terminal base / terminal unit on the DIN rail, mount the Processor / IO Module.



Press the Processor / IO Module into the terminal base / terminal unit until it locks in place.

The demounting is carried out in a reversed order. Press above and below, then remove the Processor / IO Module.



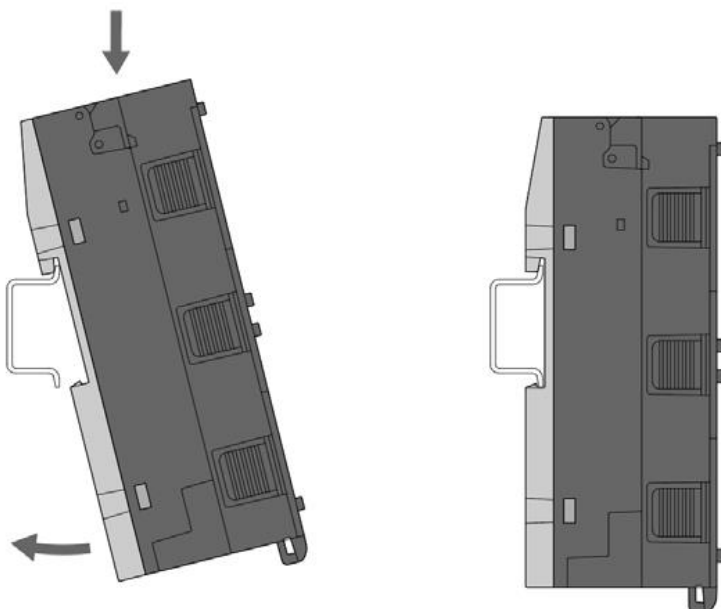
Note: Unused slots for communication modules are not protected against accidental physical contact.

Unused slots for communication modules must be covered with dummy communication to achieve IP20 rating.

I/O bus connectors must not be touched during operation.

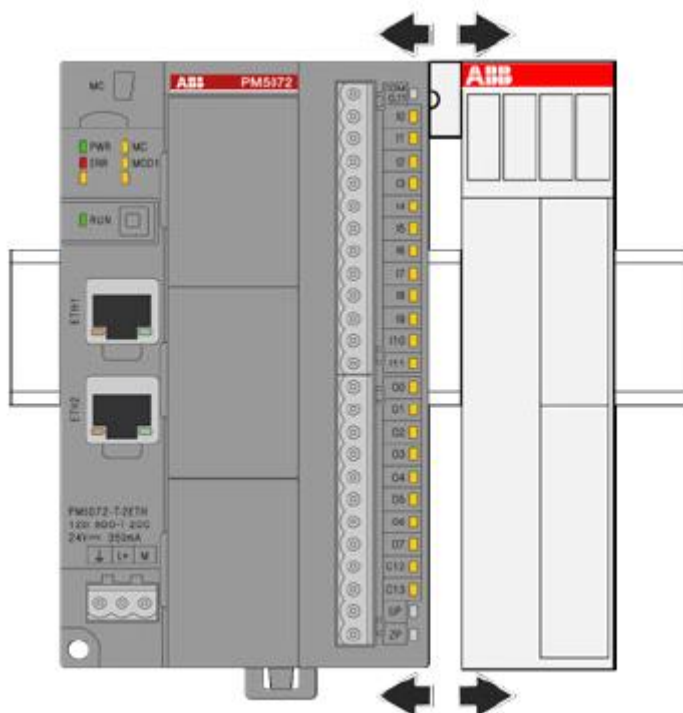
2.4.1.4 Mounting and deunmounting eCo V3 processor (CPU) and eCo S500 IO modules

Mounting a processor module on a DIN rail

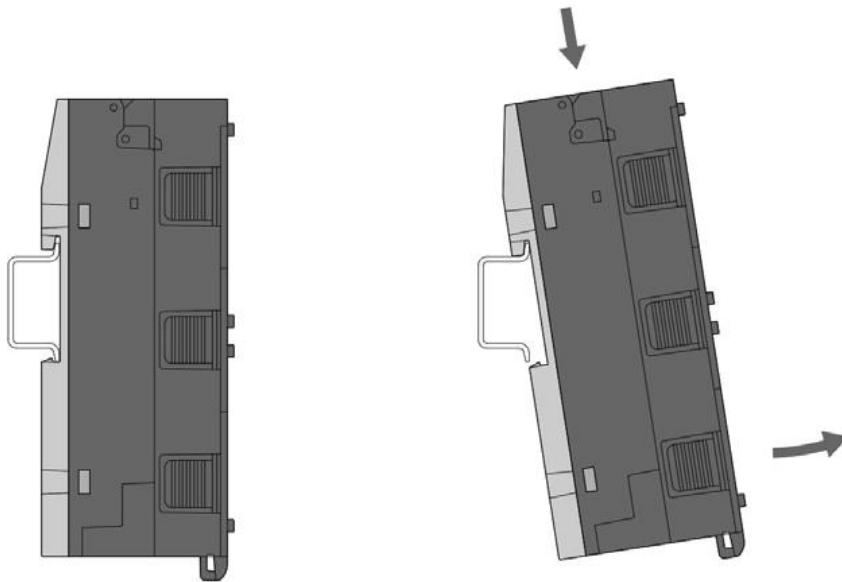


Demounting a processor module mounted on a DIN rail

1. Remove I/O modules if one is connected.

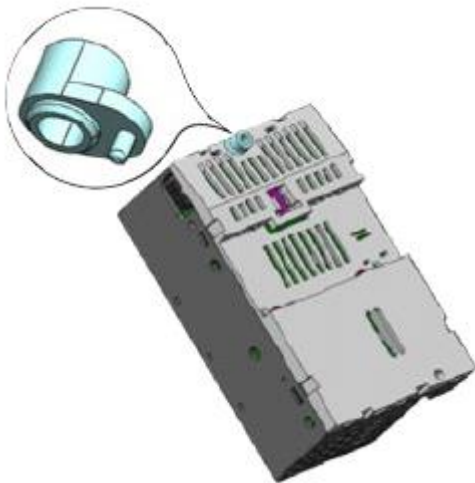


2. While pressing down processor module pull it away from DIN rail.

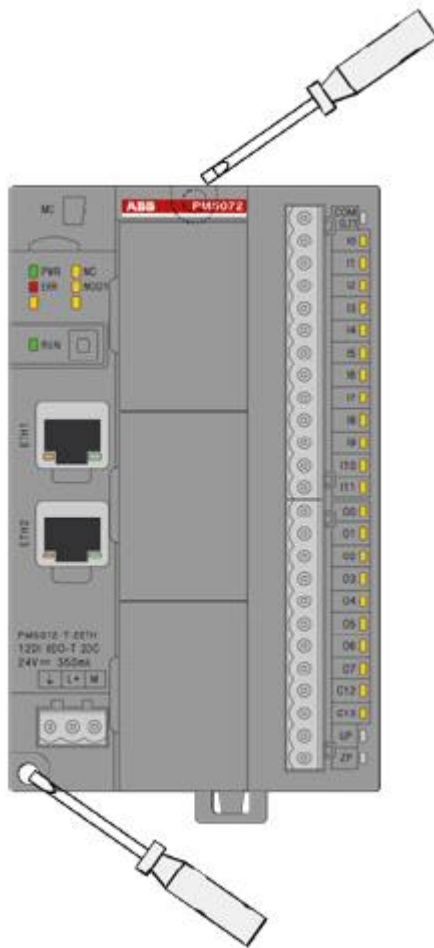


Mounting a processor module on a metal plate

1. Snap in the TA543 at the back side of the processor module

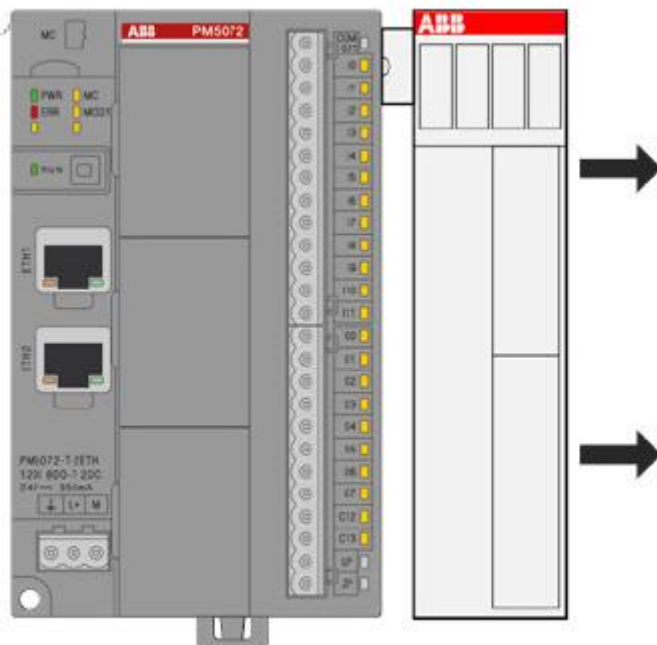


2. Fasten the processor module with two screws (max. diameter: 4 mm) to the metal plate.

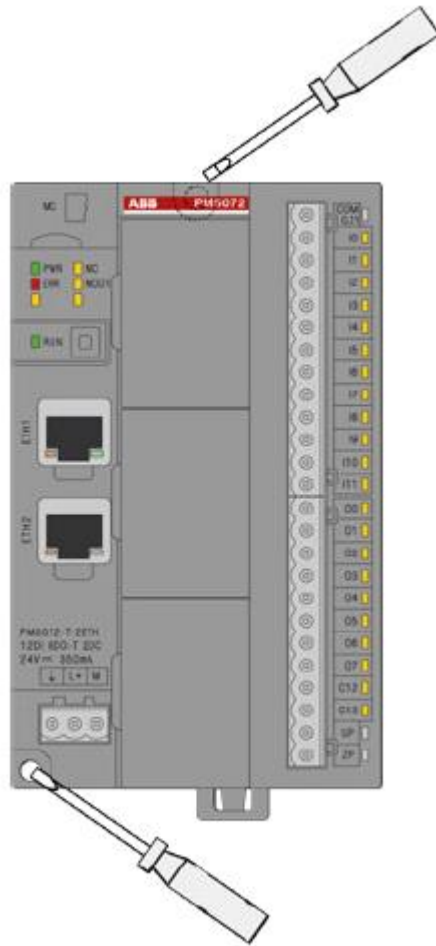


Demounting a processor module mounted on a metal plate

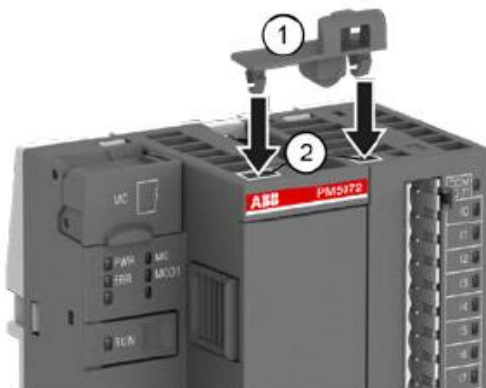
1. Remove I/O modules if connected.



2. Remove the 2 screws.



Mounting of TA5301-CFA



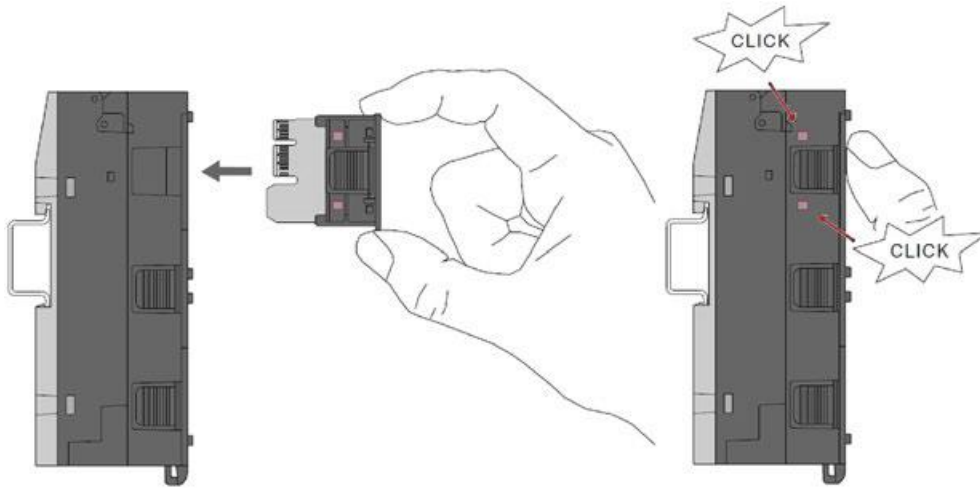
1. TA5301-CFA cable fixing accessory
2. openings on the PM50x2 processor module

a. Insert the TA5301-CFA cable fixing accessory into the two openings on the PM50x2 processor module marked white in the figure.

Mounting and demounting option boards

Inserting the option board

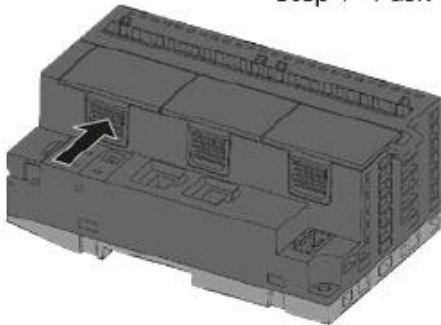
After mounting the PM50x2 processor module on the DIN rail, mount the option board.



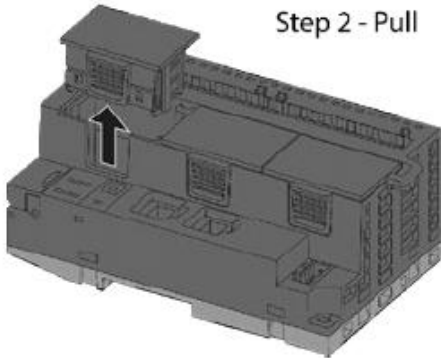
Press the option board TA51xx (or TA5300-CVR) into the slot of the processor module PM50x2 until it locks in place.

Removing the option board

Step 1 - Push



Step 2 - Pull



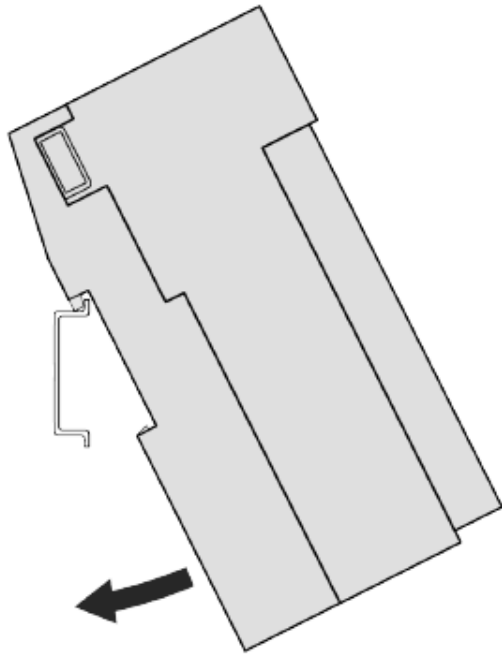
1. Push the option board on the side to release the lock.
2. At the same time, pull the option board out of the slot.

Mounting and demounting of S500-eCo I/O modules

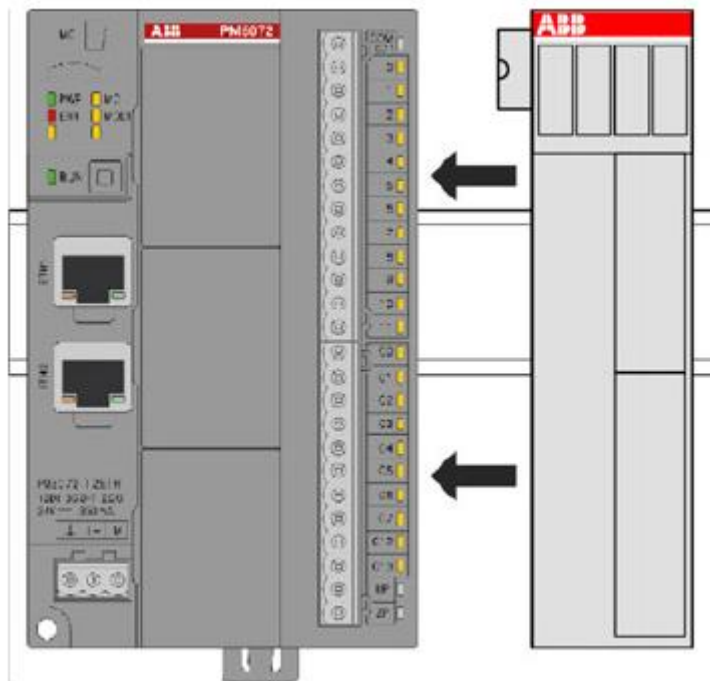
S500-eCo I/O modules can be mounted either on a DIN rail or with screws on a metal plate.

Mounting I/O modules on a DIN rail

1. Mount I/O module at the top of the DIN rail, then snap it in below.

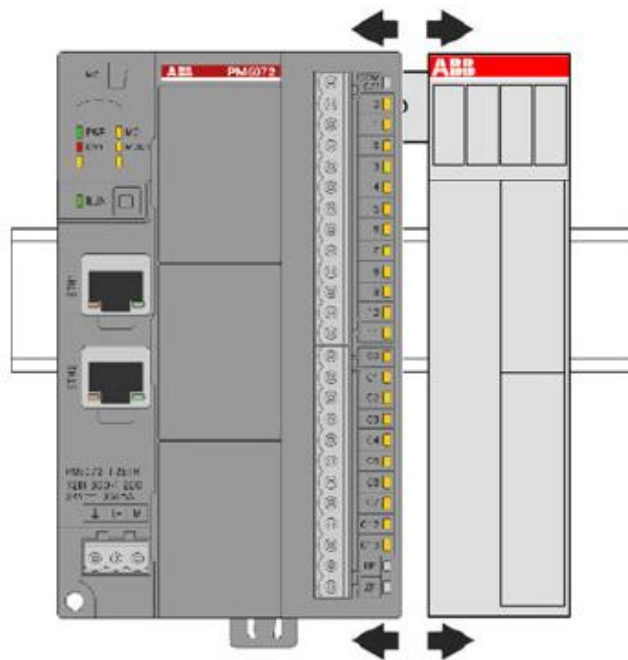


2. Attach I/O module by hand to another module. The serial I/O bus is connected automatically.

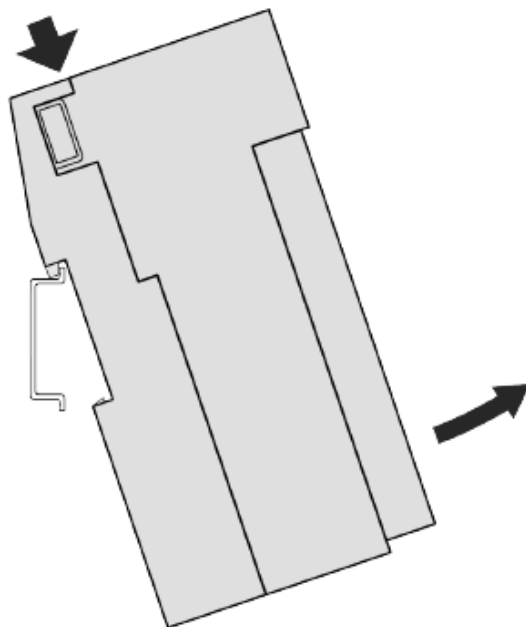


Demounting I/O modules mounted on a DIN rail

1. Remove I/O module by hand if connected

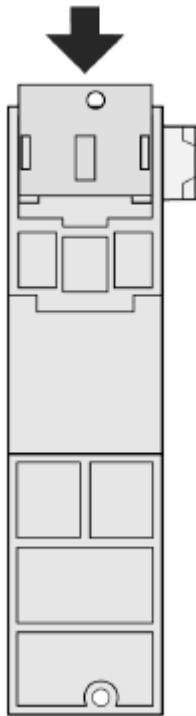


2. While pressing down I/O module pull it away from DIN rail.

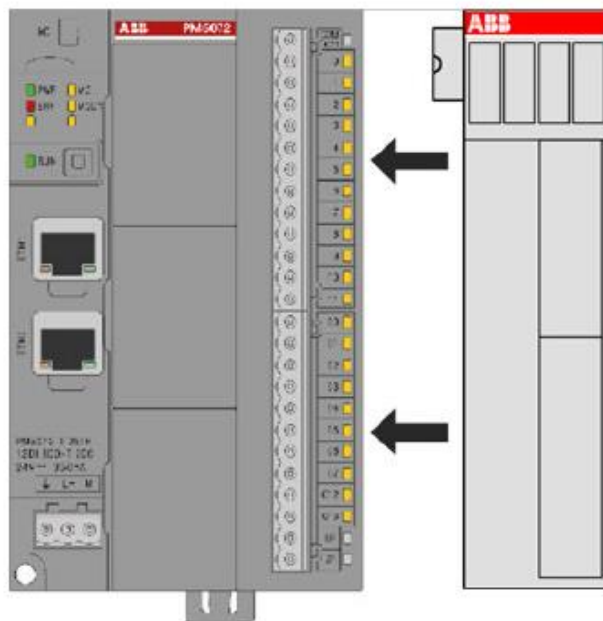


Mounting I/O modules on a metal plate

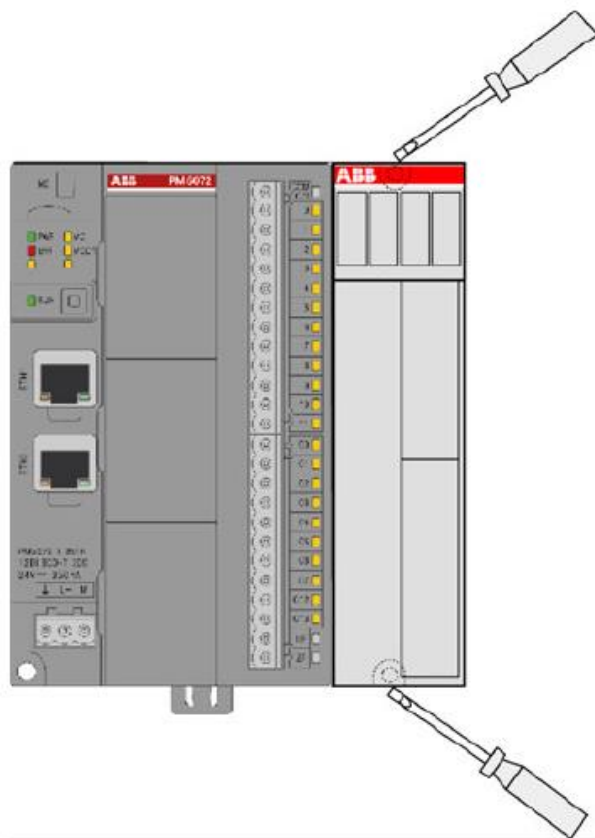
1. Snap in the TA566 at the back side of the I/O module



2. Attach the I/O module by hand to an other module. The serial I/O bus is connected automatically.

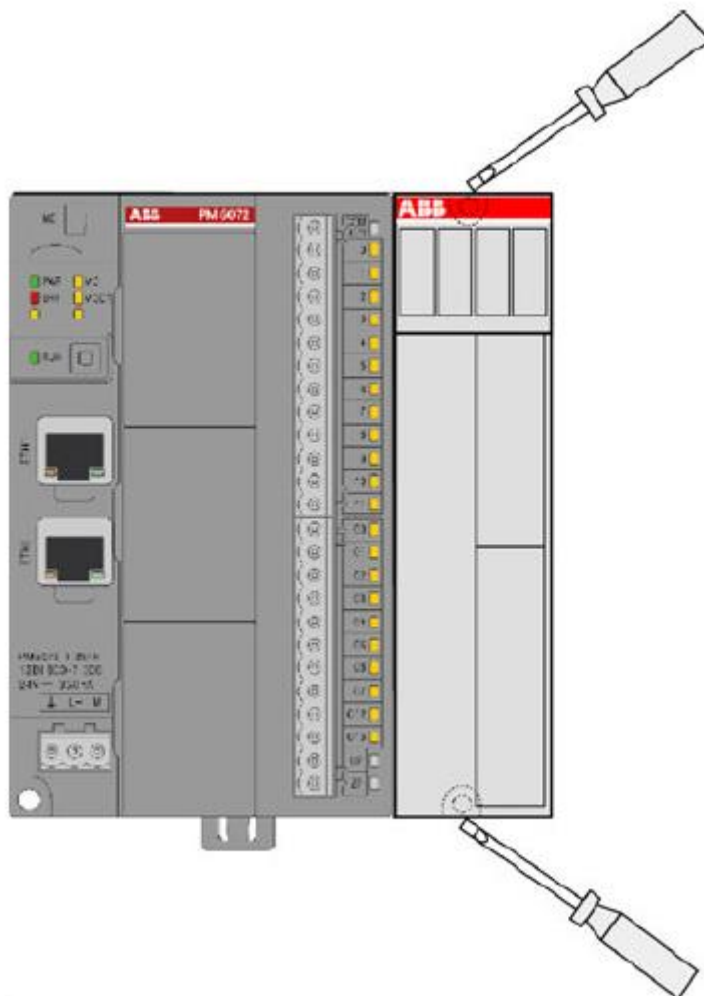


3. Fasten the I/O module with two screws (max. diameter: 4 mm) to the metal plate

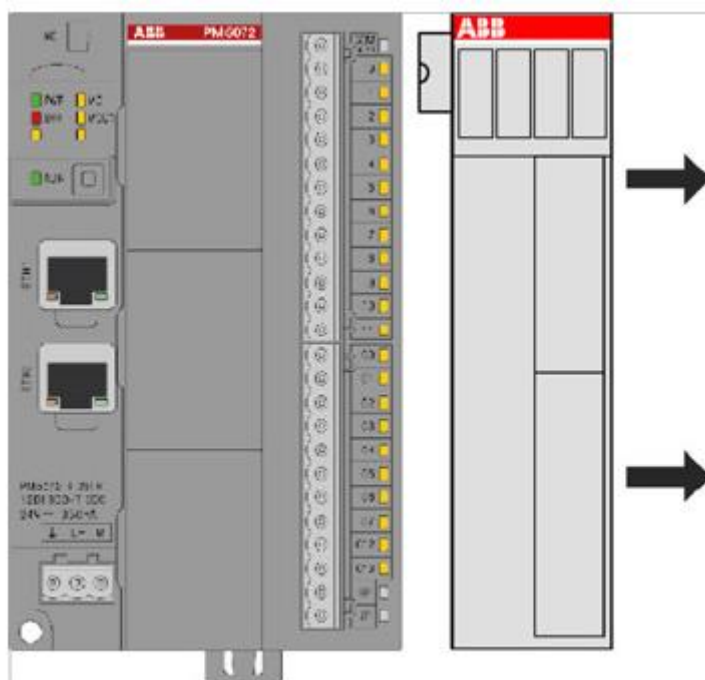


Demounting I/O modules mounted on a metal plate

1. Remove the 2 screws.



2. Remove the I/O module from the connected module by hand.

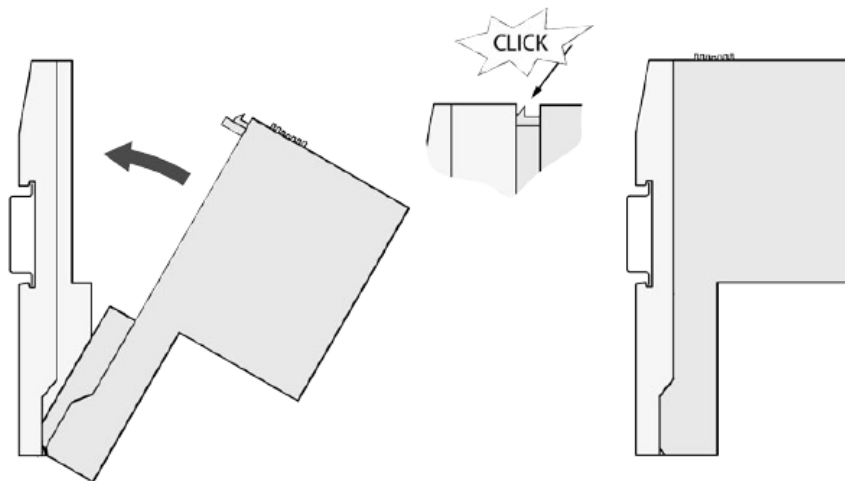


2.4.1.5 Mounting and demounting the communication modules (CMxxx-yyyy)

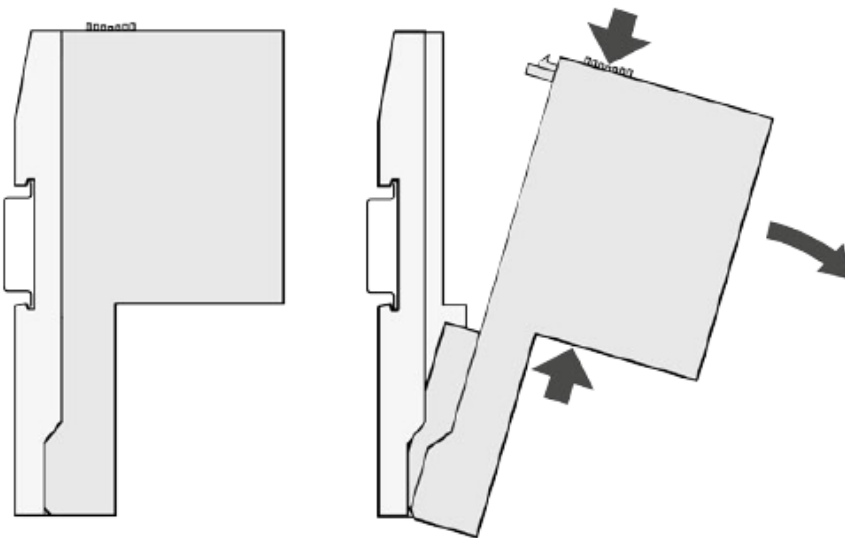
Communication Modules are mounted on the left side of the processor module on the same terminal base. The electrical connection is established automatically when mounting the communication module.

After mounting the terminal base, mount the communication modules.

1. First insert the bottom nose of the Communication Module into the dedicated holes of the Terminal Base. Then, rotate the Communication Module on the dedicated Terminal Base slot until it is locked in place.



2. The demounting is carried out in a reversed order.
3. Press above and below, then rotate the Communication Module and remove it.



2.5 Electrical connection

The PLC is equipped with a switch power supply for internal circuit. Compared with ordinary power supply, the PLC power supply has the higher stability and interference immunity. A number of PLC products provide 24 V DC stabilized voltage supply to meet external sensors.



Note: For detailed updated information such as technical data of your mounted devices (AC500 product family) refer to the latest Automation Builder help file hardware device description of the appropriate device.

2.5.1 Power supply for processor modules



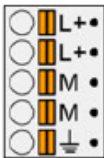
V3 Processor (CPU) power supply

The supply voltage of 24 VDC is connected to a removable 5-pin terminal block on the terminal base. L+/M exist twice and there for possible to feed e.g., external sensors (up to 8 A max. with 1.5 mm² conductor) via these terminals.

CPU power supply: AC500 logic controller power supply is provided through terminals L+ /M.

IO module power supply: S500 IO modules process power supply is provided through terminals UP / ZP.





For specific devices electrical connection data refer to the respective device help file from Automation Builder.

Pin Assignment	Label	Function	Description
 Terminal block removed	L+	+24 VDC	Positive pin of the power supply voltage
	L+	+24 VDC	Positive pin of the power supply voltage
	M	0 V	Negative pin of the power supply voltage
	M	0 V	Negative pin of the power supply voltage
		FE	Functional earth
 Terminal block inserted			

As system power supply for AC500/S500, the wide ABB CP series 24 V power supply models can be used. There are different ranges within the CP-series for all kinds of requirements and budgets: CP-C.1, CP-E, CP-S.1 and CP-D.

V3 eCo Processor (CPU) power supply

The processor modules PM50x2 can be connected to the 24 V DC supply voltage via a removable 3-pin spring terminal block or a 3-pin screw terminal block.



3-pin spring terminal block		3-pin screw terminal block		
				
Pin Assignment	Pin	Label	Function	Description
 Terminal block inserted	1		FE	Functional earth
	2	L+	+24 V DC	Positive pin of the power supply voltage
	3	M	0 V	Negative pin of the power supply voltage







2.6 CPU function keys Display and LED display

AC500 V3 PLC are having a couple of function keys / display and few LED to indicate the PLC status.

For the updated information on function keys and display, please refer to latest Automation Builder help file.

2.6.1 Description of the function keys

Function Key	Description
	Run - Toggles between RUN and STOP mode. Switching into RUN mode is only possible if an error free project has been created and downloaded with Automation Builder.
	Value - Shows different state values of the processor module.




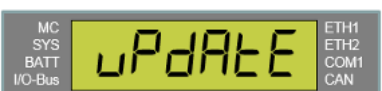
	Escape - Quits the current menu, submenu or function without saving.
	OK / Acknowledgement -Acknowledges the current value or selects a menu/submenu. Changes that have been sent to the processor module successfully are confirmed with done on the display.
	Diagnostic - Allows evaluation of error messages in detail.
	Configuration - Show/set IP configuration, PLC startup mode and Ethernet address. Enters submenus.
	Count up / navigate in submenu - Press the function key repeatedly in order to increase the value each time by 1, or navigate in submenu to previous entry Keep the function key pressed in order to count up fast.
	Count down / navigate in submenu - Press the function key repeatedly in order to decrease the value each time by 1, or navigate in submenu to next entry. Keep the function key pressed in order to count down fast.

2.6.2 Description of Display

The display shows the status of the PLC and helps while navigating to different configuration using the function keys on the processor module. For detailed information, please refer the latest Automation Builder help file.






Below are the few commonly showed display information and its meaning.

2.6.2.1 Start-up procedure of a new PLC from factory

State	Description	Display
0	Display on system start (power on).	
1	PLC is in boot mode.	
2	PLC is in initialization mode.	
3	No system firmware (System Firmware) is available. Start updating the firmware using Automation Builder or memory card.	


2.6.2.2 Start-up procedure of a PLC with system firmware

State	Description	Display
-------	-------------	---------

0	Display on system start (power on).	
1	PLC is in boot mode.	
2	PLC is in initialization mode.	
3	PLC is in STOP mode or RUN mode depending on the PLC mode and the downloaded project.	 

2.6.3 Other common display codes

Other than above, the below display is also possible to be shown by the PLC.

Description	Display
Text is shown if no communication between CPU and display is possible due to very high CPU load (e.g., endless loop in user program and not activated task watchdog).	

2.6.4 Description of LED

LED	State
PWR (Power LED)	When on and green -> the PLC is having 24 V DC present.
RUN (Run LED)	When on and green -> the PLC is in RUN mode
ERR (Error LED)	When on and red -> PLC is having an active error.

2.7 Accessories (TA521 - Lithium battery)

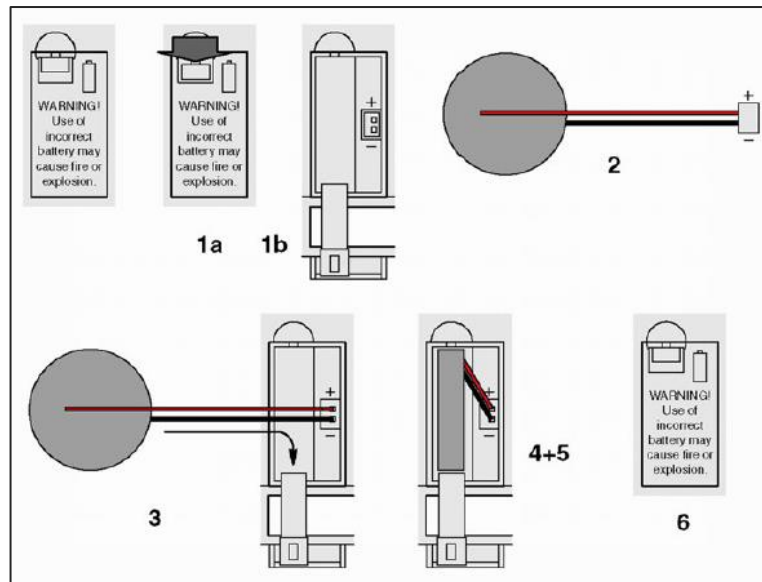
A description of all additional accessories that can be used to supplement AC500 system can be found in the Hardware PLC device description. This section only describes Lithium battery that are frequently used for system assembly, connection, and construction.

The TA521 lithium battery is the only applicable battery for the AC500 processor modules. It cannot be recharged.

The AC500 battery buffers the following data in case of "control voltage off":

Retentive variables in SRAM (static RAM) (VAR_RETAIN...END_VAR)

Date and time of the real-time clock



Open the battery compartment with the small locking mechanism, press it down and slip down the door. The door is attached to the front face of the processor module and cannot be removed.

Remove the TA521 battery from its package and hold it by the small cable.

Insert the battery connector into the small connector port of the compartment. The connector is keyed to find the correct polarity (red = plus-pole = above).

Insert first the cable and then the battery into the compartment, push it until it reaches the bottom of the compartment.

Arrange the cable in order not to inhibit the door to close.

Pull-up the door and press until the locking mechanism snaps.



Note for AC500: To ensure proper operation and to prevent data loss, the battery insertion or replacement must be always done with the system under power. Without battery and power supply there is no data buffering possible.

Note for Ac500 eCo: No Batterie is needed



Note: To prevent data losses or problems, the battery should be replaced after 3 years of utilization or at least as soon as possible after receiving the "low battery warning" indication.

Do not use a battery older than 3 years for replacement, do not keep batteries too long in stock.

2.8 Introduction to ABB PLC Licenses

ABB PLC licensing is built around the Automation Builders integrated licensing system which is designed for supporting all practical operation scenarios. Two different types of licenses has to be understood:

- **Automation Builder software licenses** are needed for the engineering tool itself and its options. During installation it has to be decided where-to-lock them to:
 - the PC,
 - a special USB dongle,
 - or a network license server.

Note: For handling details see chapter 3.2 "Software user licensing of Automation Builder"

AC500 V3 CPU Run-time licenses are needed to run the Motion Control libraries on the PLC they are installed on and allow use of features in the application program e.g. FW or downloaded libraries. They will

be locked to the CPU itself and are also handled and installed by Automation Builder (directly connected or in an offline workflow transferred via SD-card).

Note: For handling details see next chapter “CPU locked Run time license”.

In case of changes in the organization or in the engineering workflows the licenses can easily be transferred back or to where you need them. The licenses are handled with the help of CodeMeter License server which is installed on the PC/server.

You can always familiarize yourself with Automation Builder (and CPU Run-time options) using the 30 (10 for run time licenses) -day test licenses, which can be ad-hoc self-issued if not available yet or for a trial.

For further details please check and chapter 6.4 Run time license for PLC Motion Control

3 AUTOMATION BUILDER OVERVIEW

Configuration and programming of all AC500 CPU you need the engineering software suite Automation Builder. Automation Builder is available for download from <https://new.abb.com/plc/automationbuilder/platform/software>

Features:

- Programming - via Ethernet networks
- Supported Programming languages:
 - Standardized programming according to five of IEC 61131-programming languages;
 - (Structured Text (ST),
 - Function Block Diagram (FBD),
 - Instruction List (IL),
 - Ladder Diagram (LD),
 - Sequential Function Chart (SFC),
 - Two additional nonstandard programming languages are supported:
 - graphical function chart (CFC),
 - Application programming in C/C++
- Debugging and commissioning tools
- Online diagnosis
- Debugging functions for the program test: Single step, Single cycle, Breakpoint
- Offline simulation - simulate commands without PLC being connected
- Sampling trace - timing diagrams for process variables
- Assistants for the Configuration of the communication interface modules (for PROFINET, EtherCAT, CANopen, Ethernet, Modbus)
- Export and import interfaces for devices, signals, applications, visualization, etc.
- Usability Features
- Multi-user support and project compare
- Project scripting
- Recipe management and watch lists

Visualization (PLC based HMI for web and local debugging) Comprehensive libraries

- Export and import interfaces for devices, signals, applications, visualization, etc.
- Multi-user support and project compare
- Project scripting

3.1 Software installation



Note: For latest updated information on AC500 products, please always refer to latest Automation Builder online help file.

3.1.1 Preconditions

3.1.1.1 System requirements

For the current system requirements of the Automation Builder, please refer to the latest published **release notes**.



Note: The latest published release notes can also be found in the Automation Builder menu under “Help -> Automation Builder Release Notes”.

For Automation Builder 2.x the system requirements are:

- 1 gigahertz (GHz) or faster 32-bit (x86) or 64-bit (x64) processor
- 8 GB RAM
- 5-18 GB available hard disk space depending on the selected feature set (in addition to Operating System (OS) and other applications)

Supported operating systems:

- Windows 10 (32/64 Bit) Professional / Enterprise
- Windows Server 2012 R2 64 bit (all devices have to be directly accessible by the server; requires enabled .NET Framework 3.5)
- Windows Server 2019 (all devices have to be directly accessible by the server; requires enabled .NET Framework 3.5)



Note: Windows 7 is no longer supported.

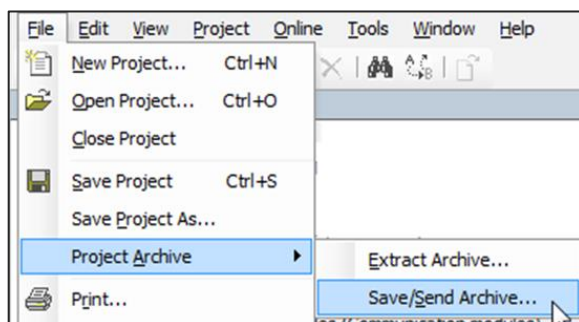
Network licenses are required for use of Automation Builder on Windows Server operating systems.

3.1.1.2 Creating project archives before an upgrade installation

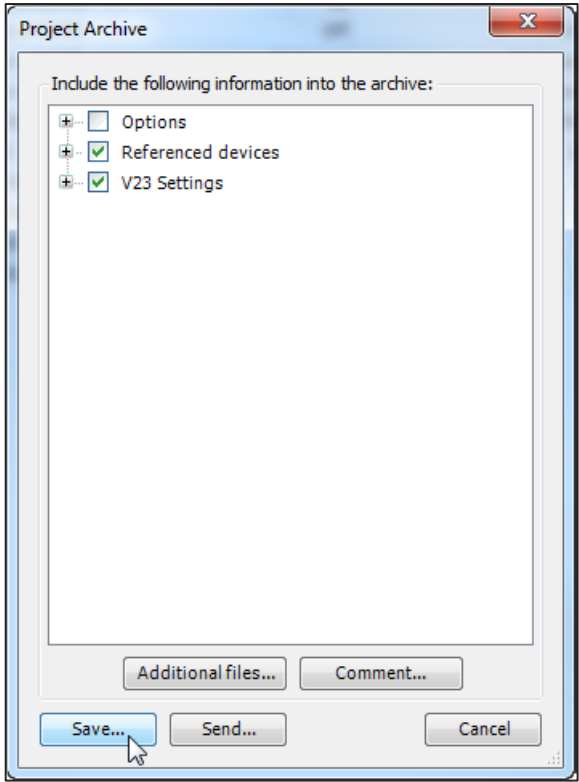
To ensure a smooth upgrade of your projects created with former Automation Builder versions its advised to create project archives of these projects.

Creating project archives

In the Automation Builder menu select “File ➔ Project Archive ➔ Save/Send Archive...” to create an archive which includes all ABB and third-party devices.



Select the information you want to include

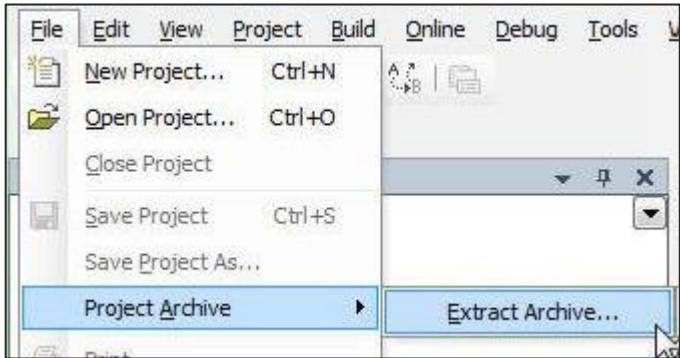


Enter a file name and location for your archive.

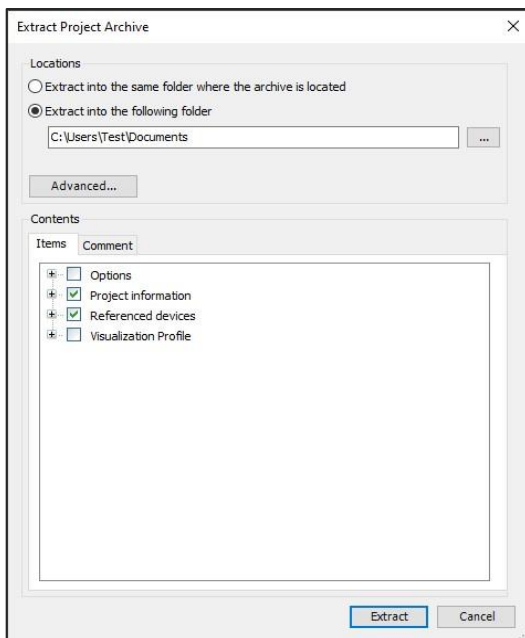


Opening archived projects

In the Automation Builder menu select “File -> Project Archive -> Extract Archive...” to open an archived project. Otherwise, by just opening the *. project files some devices (e.g. CS31 Bus) might be missing. Also, this will provide the third-party device description to Automation Builder automatically. So, it´s not necessary to install all these third-party device descriptions manually.



Select which content you want to extract from your archive.



3.1.1.3 Check internet connection and firewall settings

Please check that you have a working internet connection.

Best practice is to open an internet site in parallel and leave that open during download or licensing process.

In case of errors please check if your firewall has maybe blocked Automation Builder and adapt the firewall settings.




Note: If you have any problems with your internet connection or proxy authentication or firewall settings, then choose the offline installation and licensing.

3.1.2 Online Installation

Go to <https://new.abb.com/plc/automationbuilder/platform/software> to access the download page of Automation Builder.

In the “Latest Automation Builder” section, select “Automation Builder x.x. Download” (x.x. = latest version). This downloads the installer on your computer.



HOME → OFFERINGS → PLC AUTOMATION → AUTOMATION BUILDER → PLATFORM → SOFTWARE

Automation Builder software download

Automation Builder is available in Basic, Standard and Premium editions meeting the needs of small projects and managing the challenges of many and large projects for OEM and system integrators.

Start working immediately: After installation, on the first start-up of Automation Builder you can choose from different licenses:

Free 30-day trial license – unlocking standard and premium features

Free Basic edition

Purchased standard or premium license

Licenses can be activated, removed and transferred anytime

Availability of network licenses for installation on a license server

Life-cycle support: When installing Automation Builder you can include former version profiles into your installation to maintain compatibility with projects done in former versions of Automation Builder. Alternatively you will find installation files for selected former versions in the ABB Library below.

Latest Automation Builder version (recommended): Automation Builder 2.5.0

Automation Builder 2.5 Download

Automation Builder 2.5 Release Note

Previous Automation Builder versions:

Automation Builder 2.4 Download

Automation Builder 2.3 Download

Automation Builder 2.2 Download

Automation Builder 2.1 Download

Double click on the downloaded installer -> select language for installation -> click “OK”

ABB Automation Builder

ABB

Select the language for the installation from the choices below

English (United States)

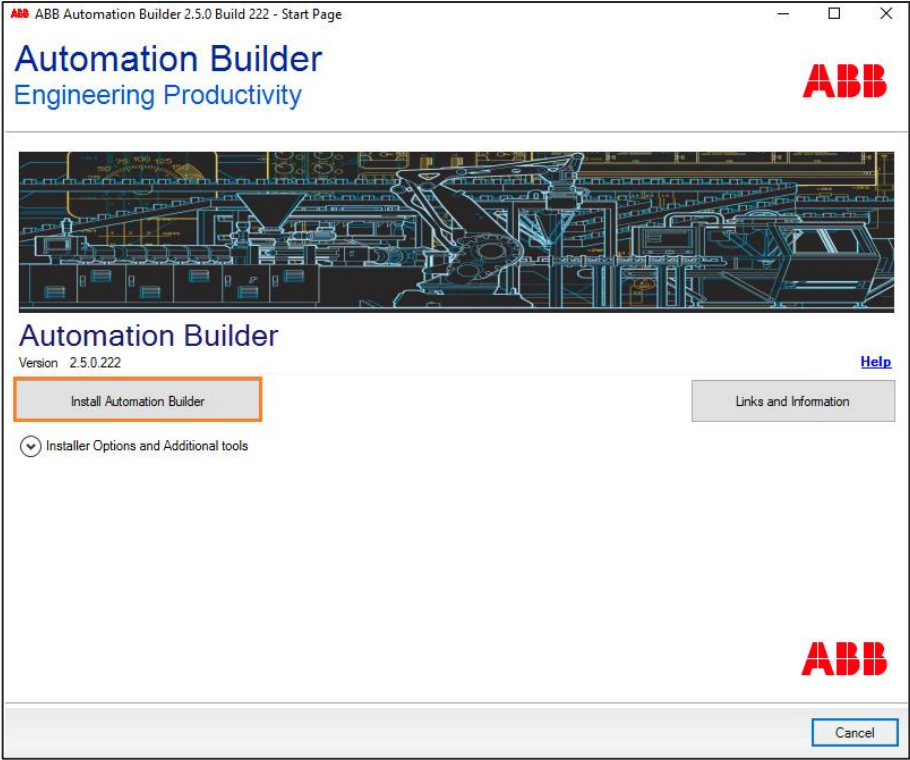
OK

Cancel

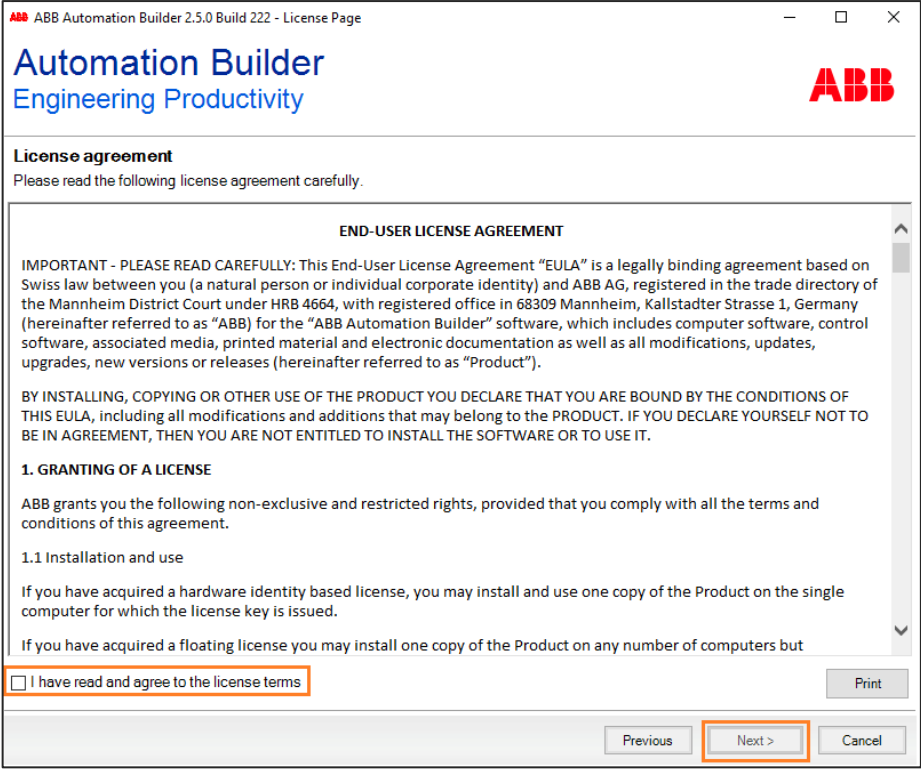
Wait for few seconds until Automation Builder installer is open as below and click on “Install Automation Builder”.

ABB Automation Builder


Please wait while the installer initializes ...



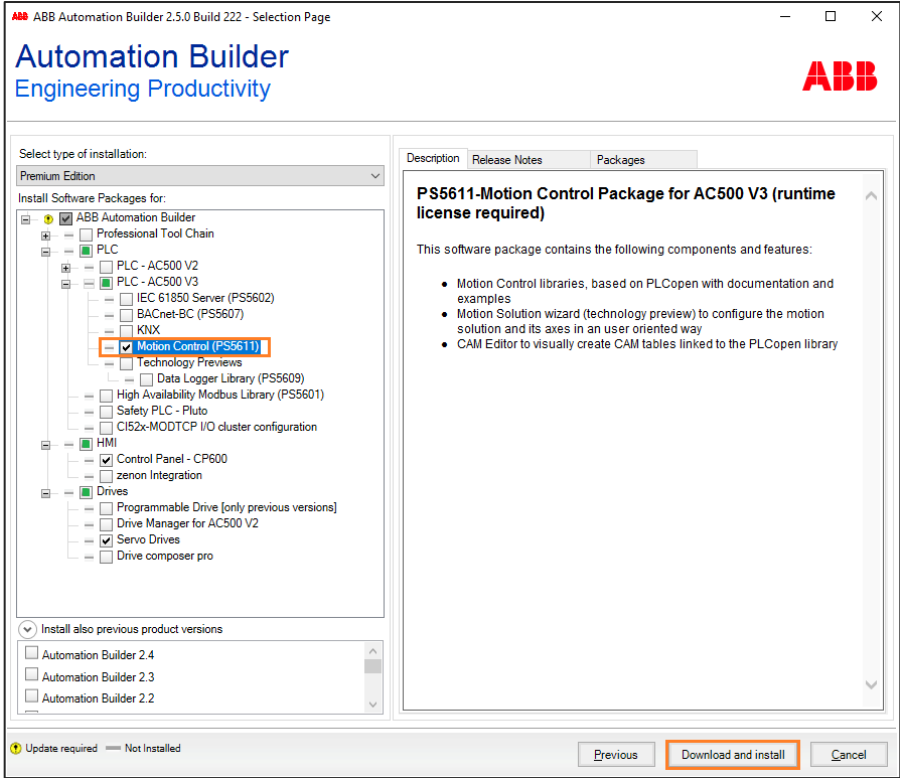
Accept the license agreement and click “Next” to proceed further




Keep the default type of installation to “Premium Edition”. Select the software packages to be installed and click “Download and install” and follow the instructions of the installation manager.



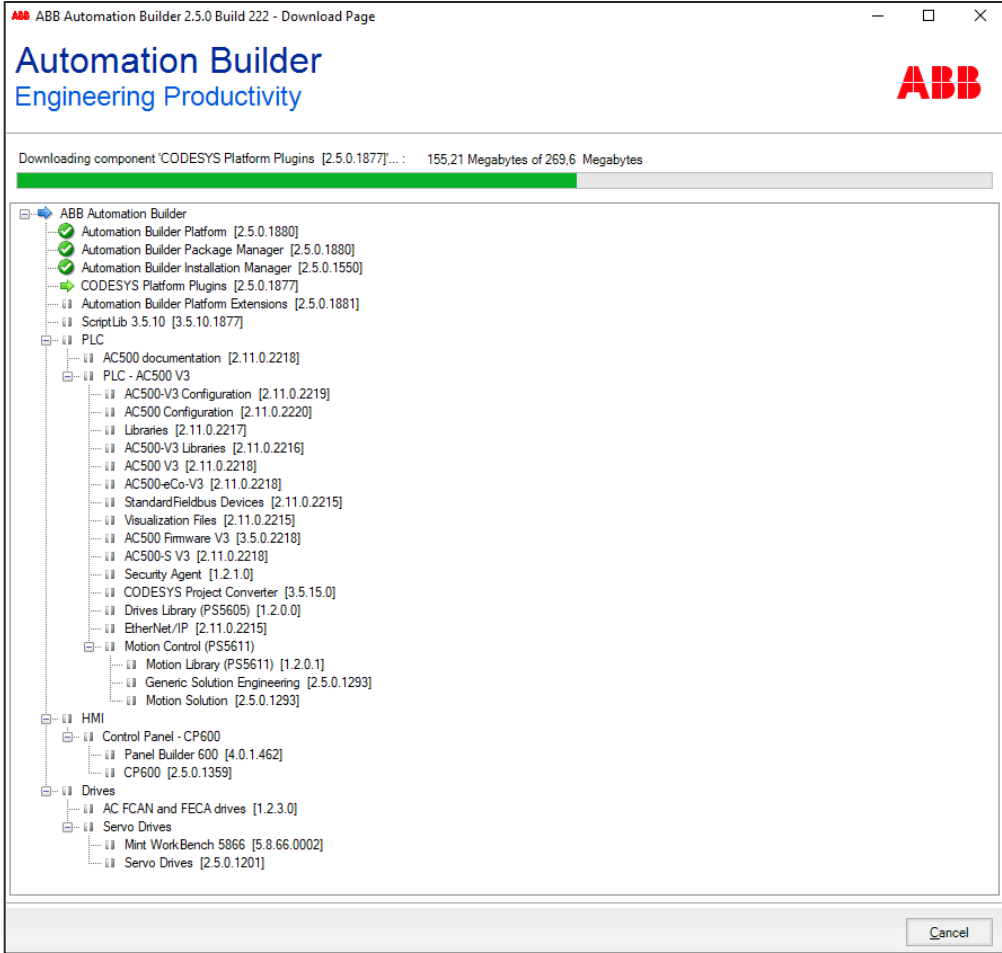
Note: – V3 Motion Control (PS5611) is not selected by default. Package path in installer-> ABB Automation Builder -> PLC -> PLC-AC500 V3 -> Motion Control (PS5611).



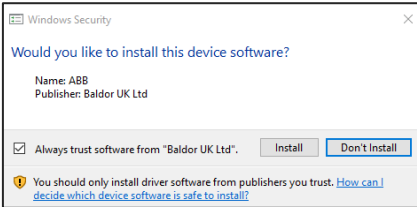
Now Automation Builder will download and install to your computer, and you can watch the download and installation progress in installation manager.



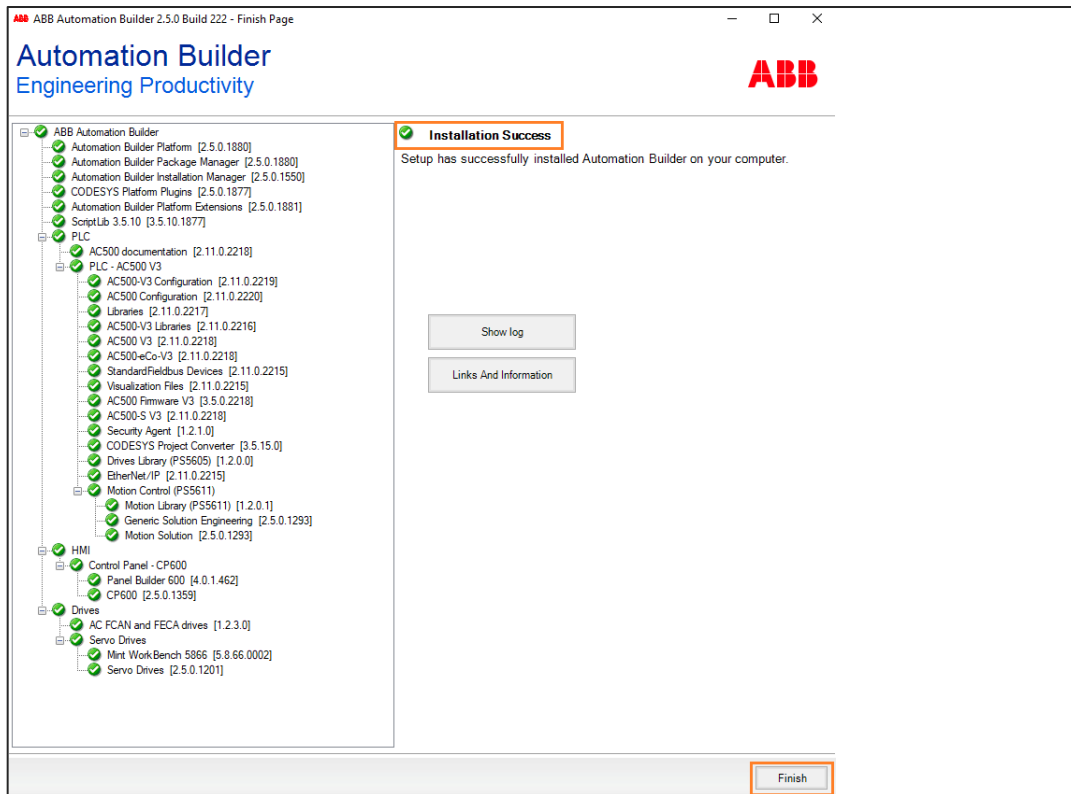
Note: – Based on the software packages selected in previous step, Automation Builder installation time varies.



Note: – You may get once or twice Windows Security popup messages during the installation while installing packages from “Servo Drives”, please click “Install” to continue the installation.



After successful installation of all packages selected, you will receive following updates on your installation manager and click “Finish” to complete the installation.




Note: – Please refer the chapter “Licensing” to get more information on activating the Automation Builder license.

3.1.3 Offline Installation

On an online computer go to <https://new.abb.com/plc/automationbuilder/platform/software> to access the download page of Automation Builder.

In the “Latest Automation Builder” section, select “Automation Builder x.x. Download” (x.x = latest version). This downloads the installer on your computer.



HOME → OFFERINGS → PLC AUTOMATION → AUTOMATION BUILDER → PLATFORM → SOFTWARE

Automation Builder software download

Automation Builder is available in Basic, Standard and Premium editions meeting the needs of small projects and managing the challenges of many and large projects for OEM and system integrators.

Start working immediately: After installation, on the first start-up of Automation Builder you can choose from different licenses:

Free 30-day trial license – unlocking standard and premium features

Free Basic edition

Purchased standard or premium license

Licenses can be activated, removed and transferred anytime

Availability of network licenses for installation on a license server

Life-cycle support: When installing Automation Builder you can include former version profiles into your installation to maintain compatibility with projects done in former versions of Automation Builder. Alternatively you will find installation files for selected former versions in the ABB Library below.

Latest Automation Builder version (recommended): Automation Builder 2.5.0

Automation Builder 2.5 Download

Automation Builder 2.5 Release Note

Previous Automation Builder versions:

Automation Builder 2.4 Download

Automation Builder 2.3 Download

Automation Builder 2.2 Download

Automation Builder 2.1 Download

Double click on the downloaded installer -> select language for installation -> click “OK”

ABB Automation Builder

ABB

Select the language for the installation from the choices below

English (United States)

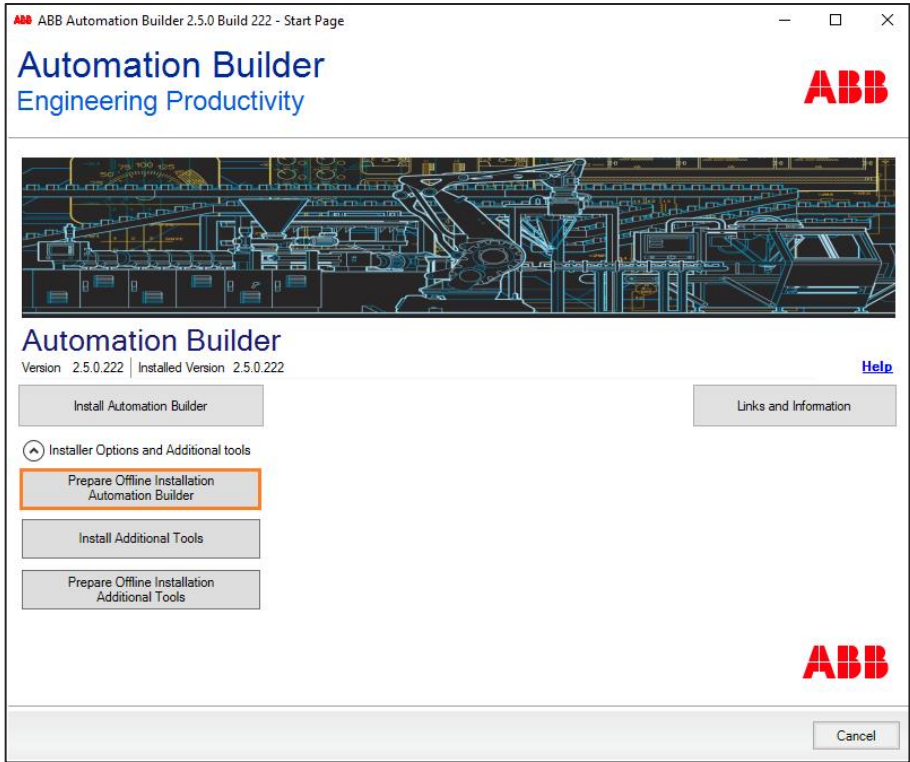
OK

Cancel

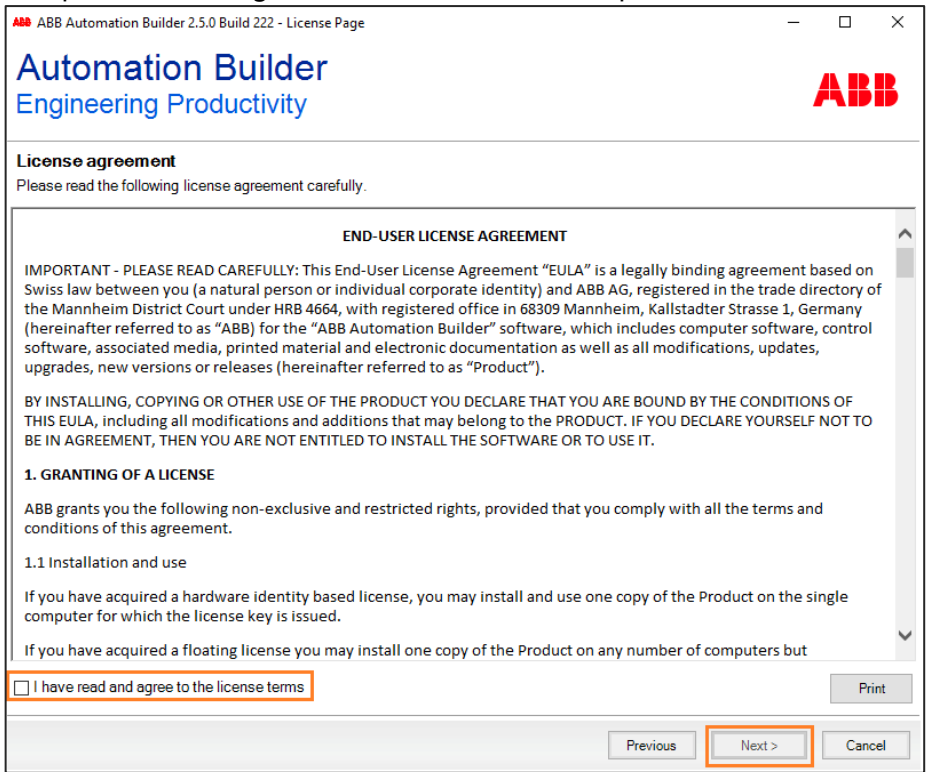
Wait for few seconds until Automation Builder installer is open as below and click on “Prepare Offline Installation Automation Builder”.

ABB Automation Builder

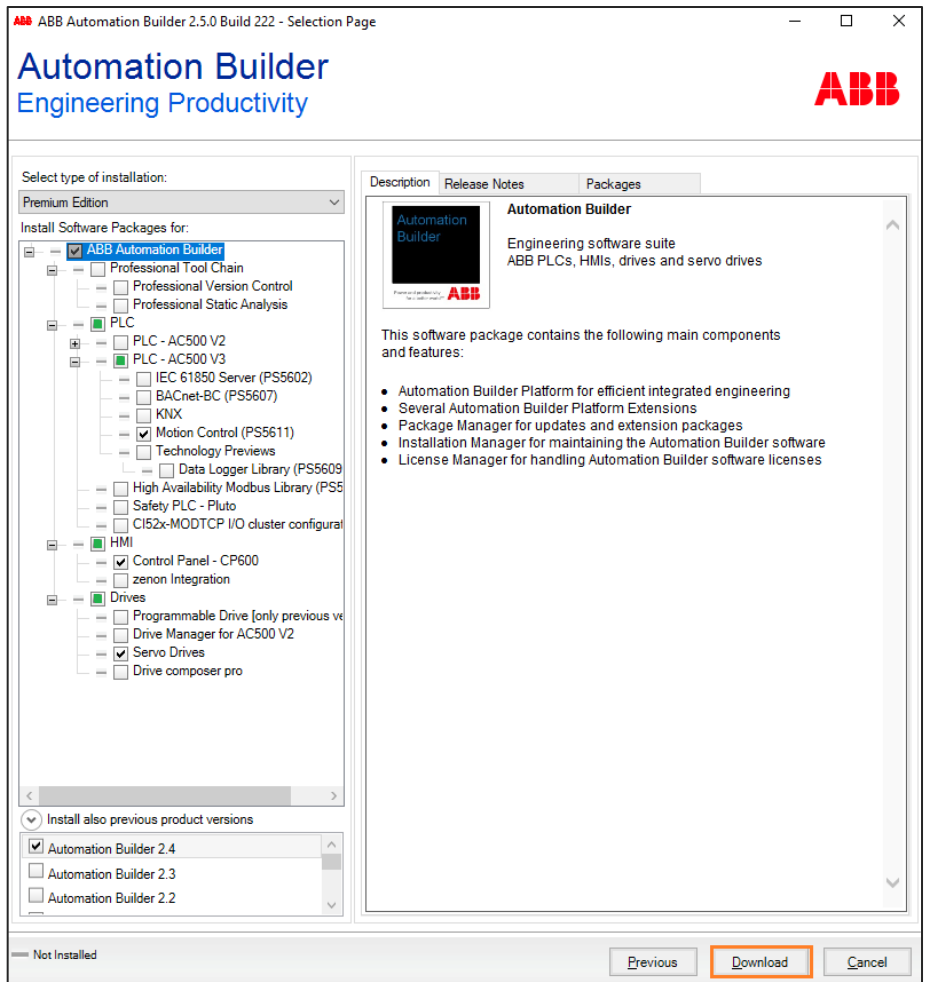
Please wait while the installer initializes ...



Accept the license agreement and click “Next” to proceed further

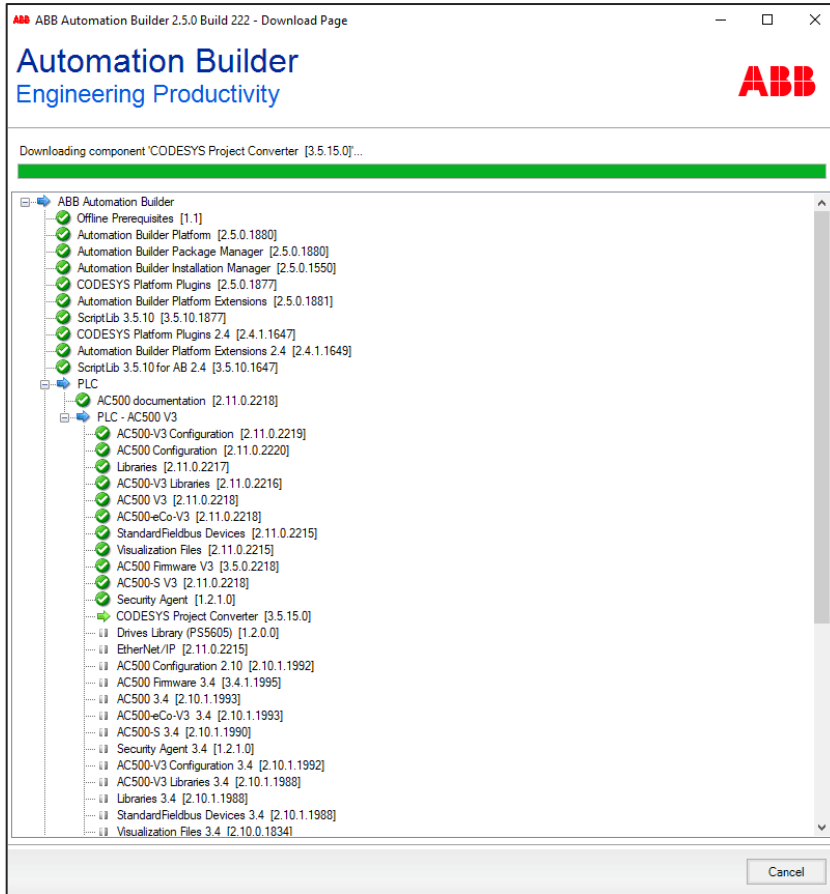


Keep the default type of installation to “Premium Edition”. Select the software packages which needs to be downloaded as offline installation package and click “Download”.

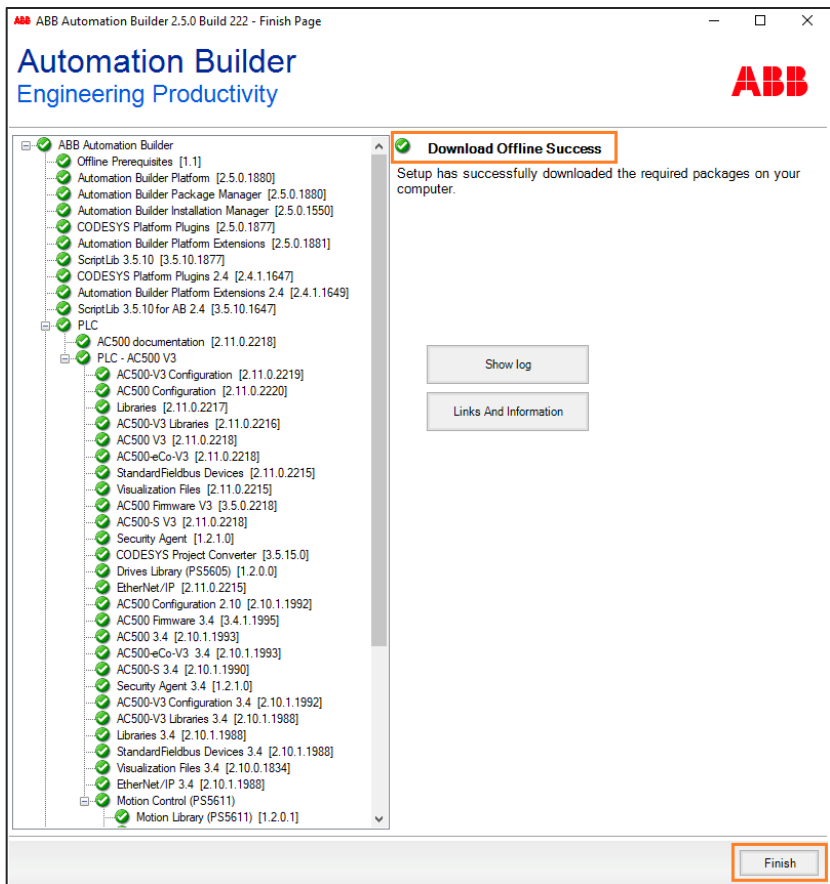


Note: – Automation Builder offline package is created with the selected software packages. Which components will be finally installed needs to be selected when performing the offline installation on the offline computer itself so it might be better to select more / all components for this offline installation package to have the choice at the installation later on.

Select a directory where the offline installation package will be saved and now Automation Builder offline installation packages will be downloaded.

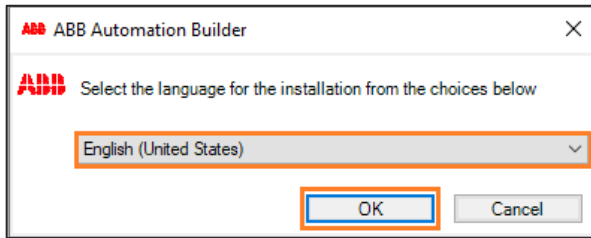


After successful download of all selected components user will get the following update on your installation manager and click “Finish” to complete the offline download process.

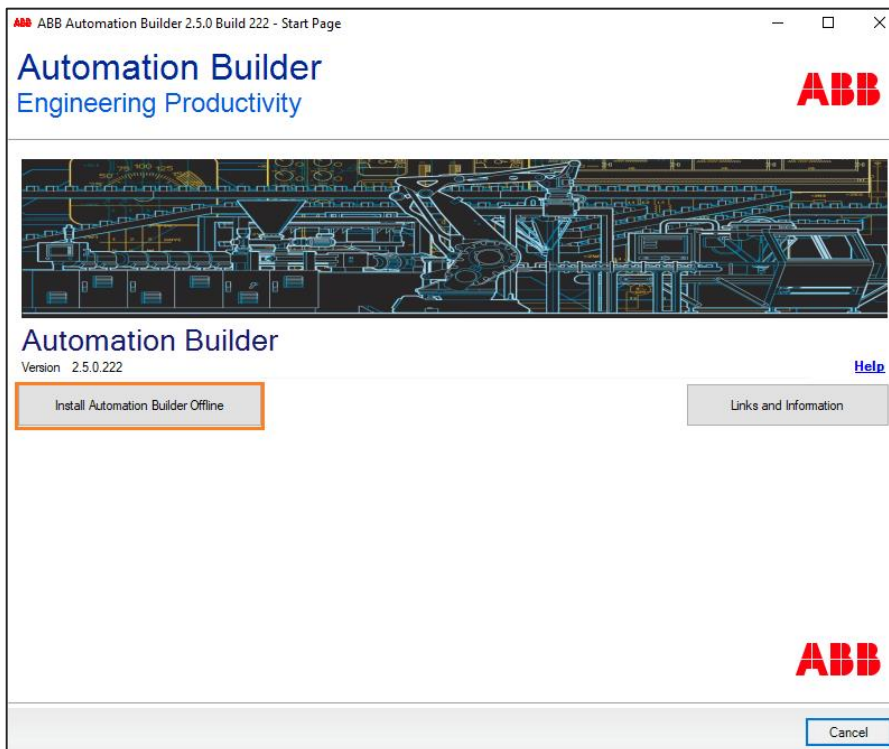
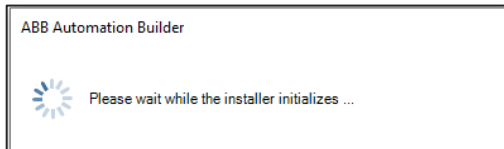


Transfer the offline installation package to the offline computer.

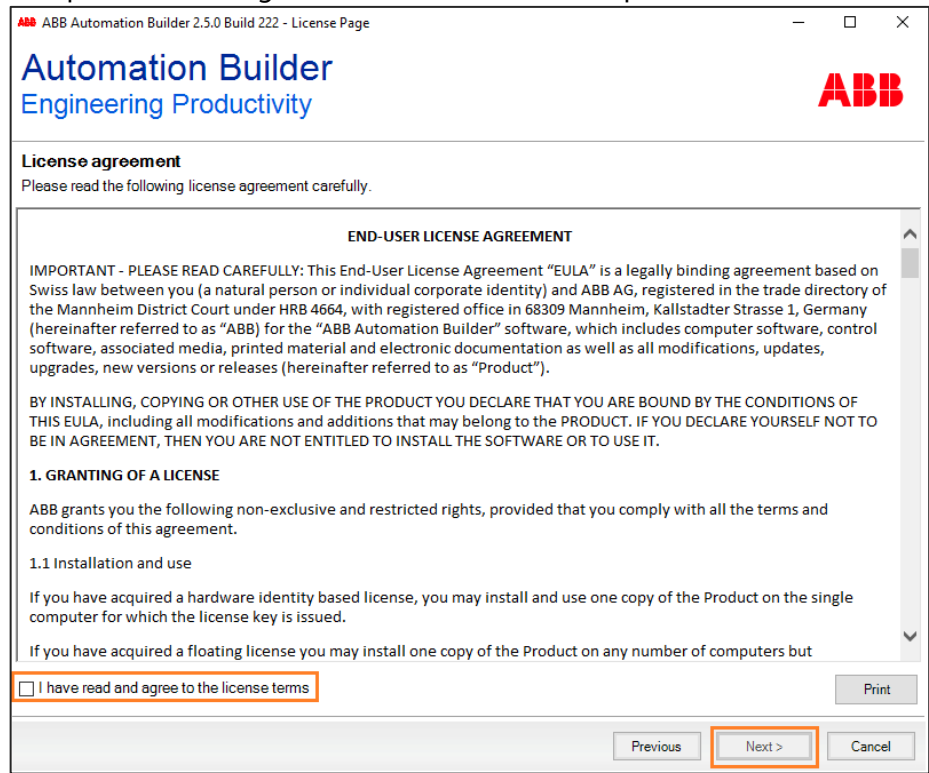
Double click on the “start_menu.exe” to start the Automation Builder installation -> select a language for installation -> click “Ok”



Wait for few seconds until Automation Builder installer is open as below and click on “Install Automation Builder Offline”.

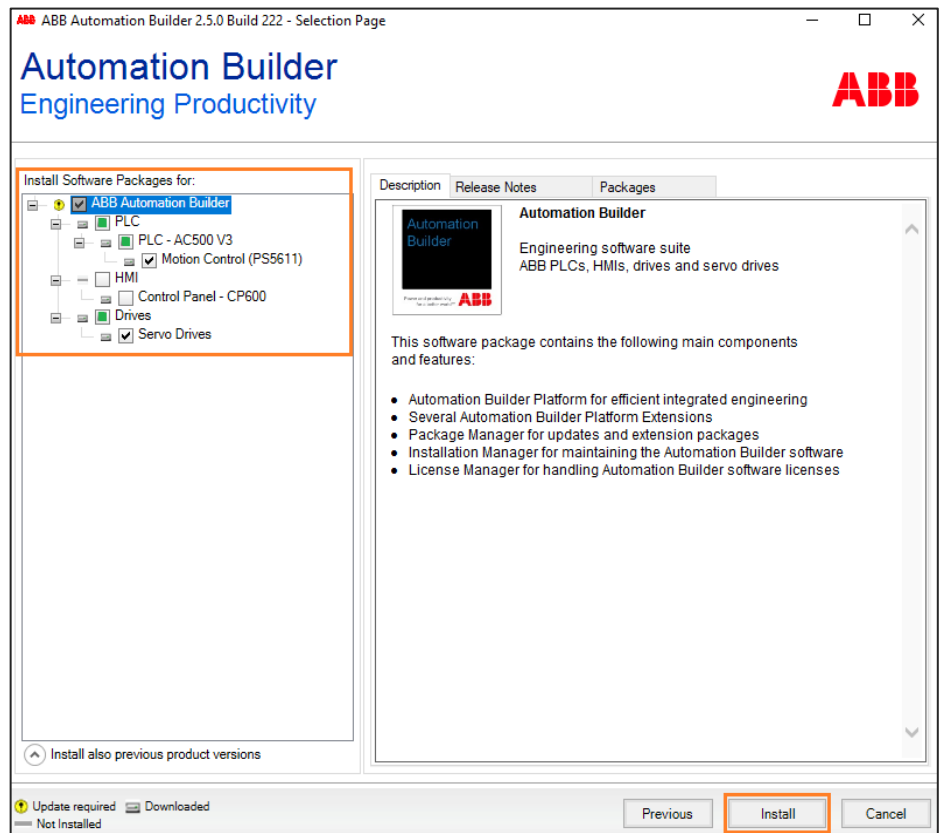


Accept the license agreement and click “Next” to proceed further




Installation manager now shows all the offline packages downloaded and user can now make the packages to be installed in offline computer.

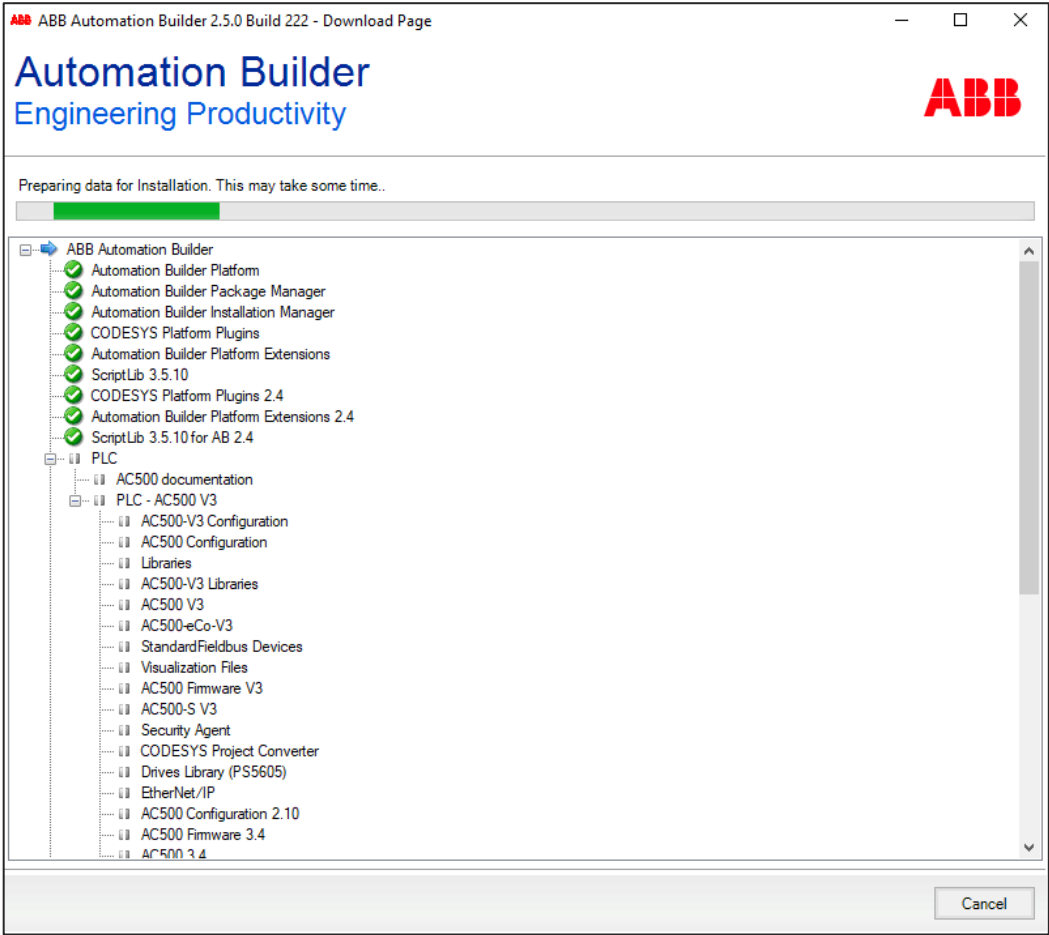
Keep the default type of installation to “Premium Edition” and select the software packages to be installed and click “Install” and follow the instructions of the installation manager.



Automation Builder will now install on the computer, and user can watch the installation progress in installation manager.

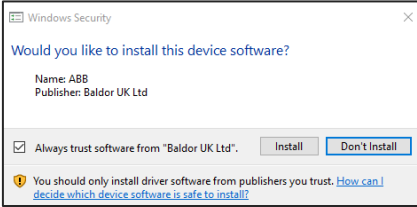


Note: – Based on the software packages selected in previous step, Automation Builder installation time varies.

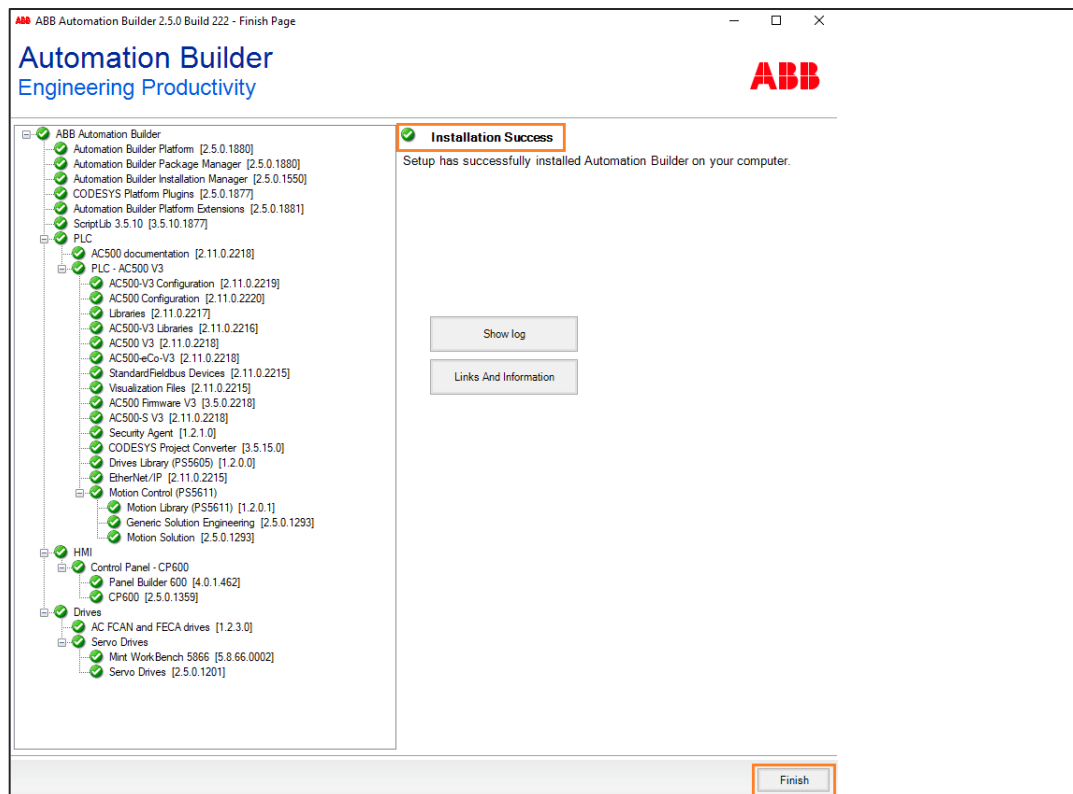




Note: – You may get once or twice Windows Security popup messages during the installation while installing packages from “Servo Drives”, please click “Install” to continue the installation.



After successful installation of all packages selected, you will receive following updates on your installation manager and click “Finish” to complete the installation.



Note: – Please refer the chapter “Licensing” to activate the Automation Builder license.

3.1.4 Installing additional tools

Sometimes the installation and activation of additional tools, e.g. OPC Server, Panel Builder 600, is needed with or without the whole installation of the Automation Builder Suite.

They can be installed separately and independent of the Automation Builder.

Various additional tools are available, e.g.

ABB License Manager

CODESYS OPC DA Server

IP Configuration Tool

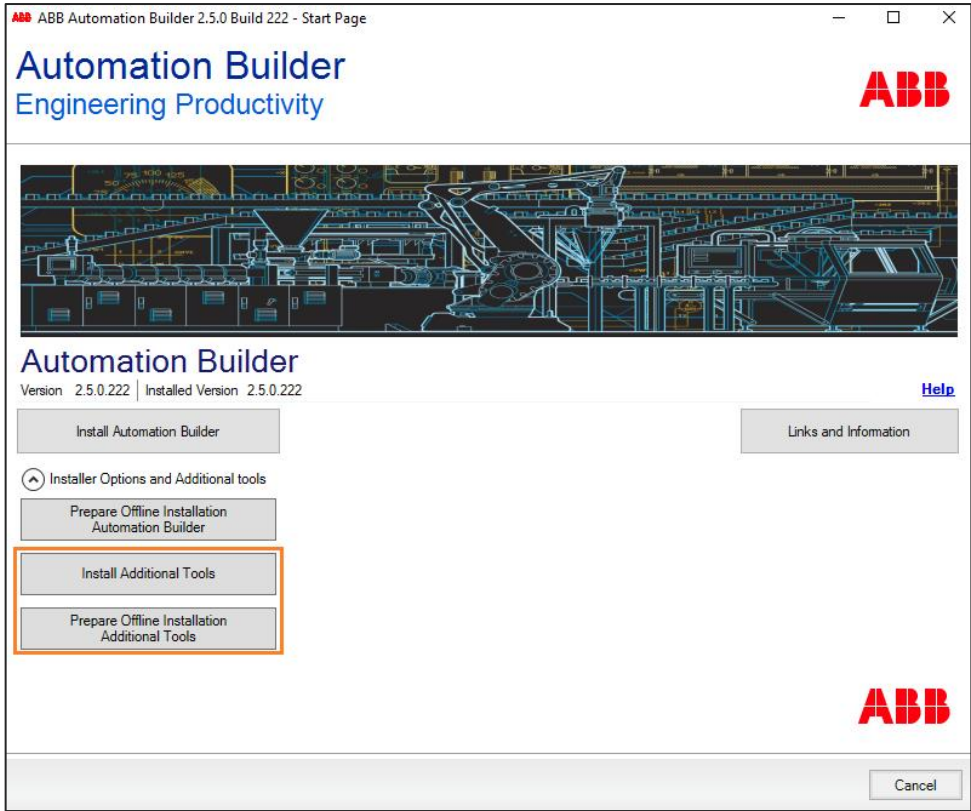
MultiOnlineChange Tool

Control Panel – CP600


Panel Builder 600 Runtime for PC

etc.....

The additional tools must be installed in the same way, as the Automation Builder installation explained in previous chapters.

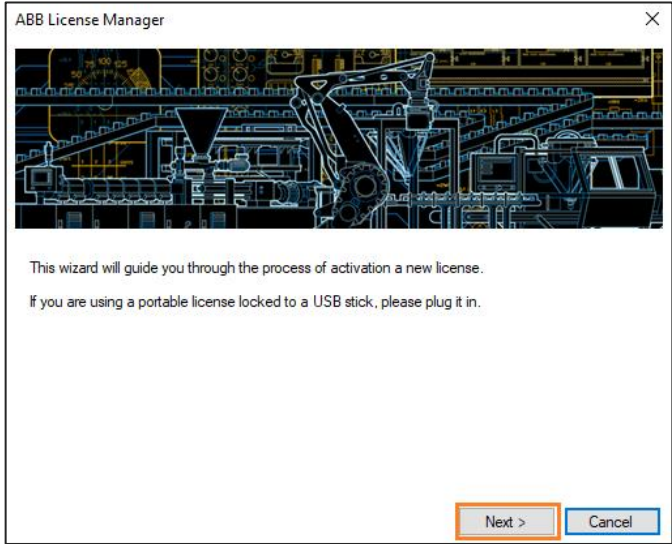


3.2 Software user licensing of Automation Builder

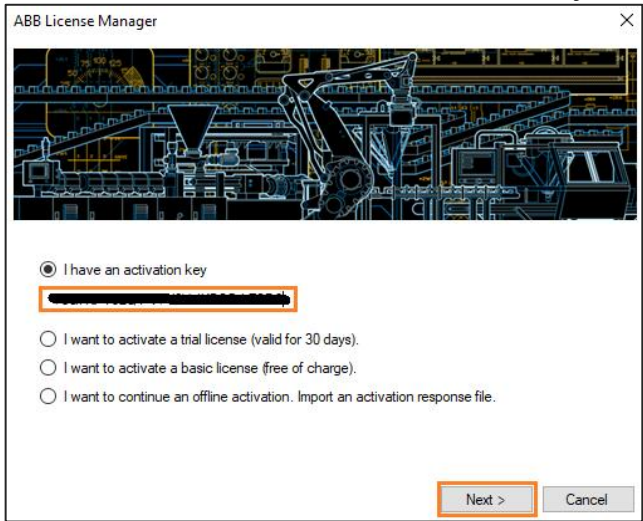


Note: For latest updated information on AC500 products, please always refer to latest Automation Builder online help file.

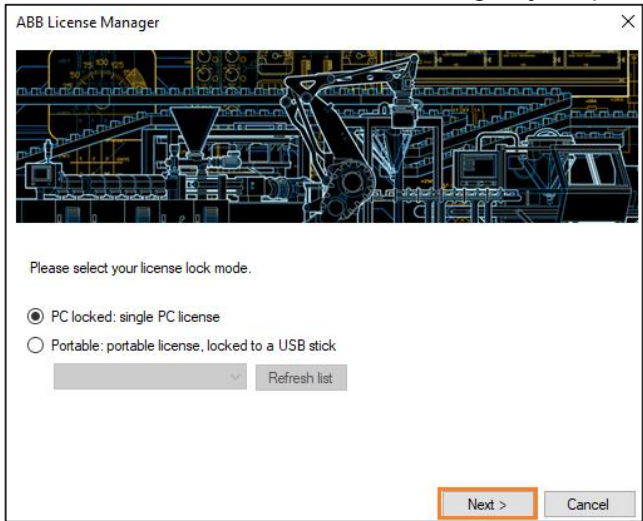
Start Automation Builder and a licensing wizard starts and guides you through the licensing procedure. Click “Next” to continue.



Enable the option “I have an activation key” and enter the activation key and click “Next”. For further information on how to receive the activation key, contact ABB sales team.



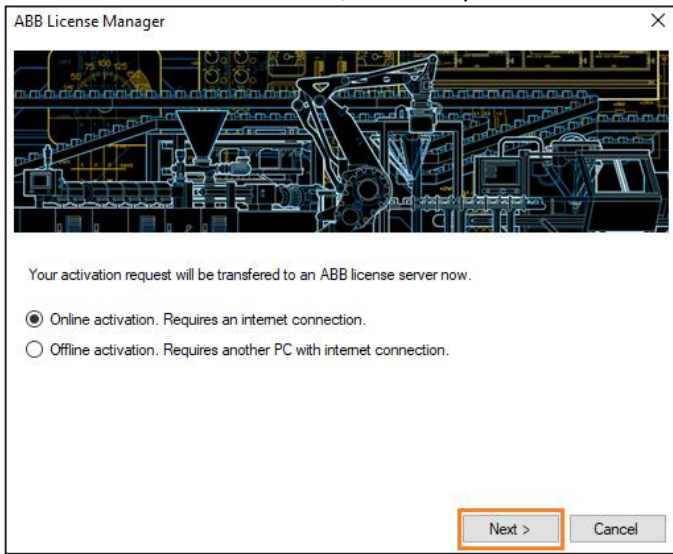
Select the license lock mode according to your purchased license and select “Next”.



Select the activation mode and click “Next”.

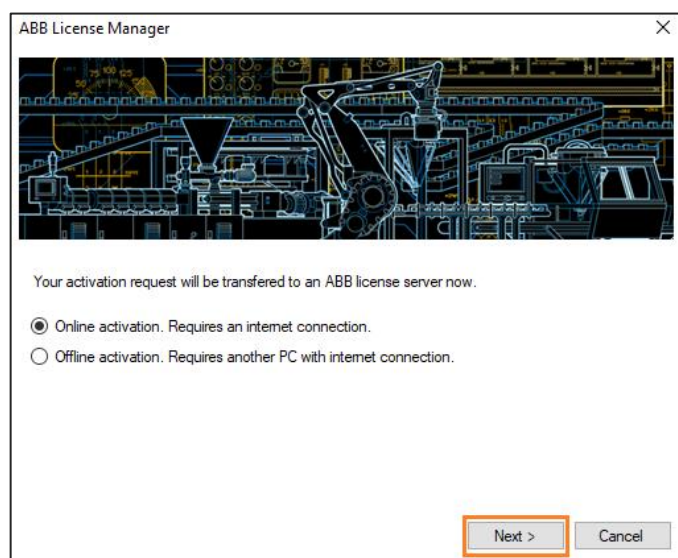
Select “Online Activation” (Refer chapter - Online Activation) if the computer has access to the internet else

Select “Offline Activation” (Refer chapter - Offline Activation)

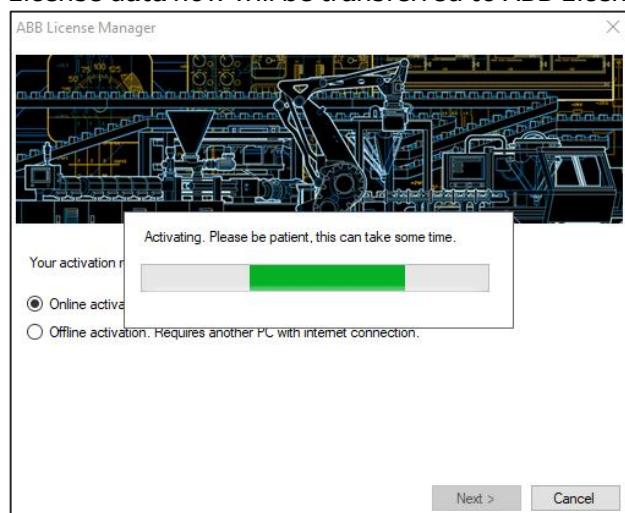


3.2.1 Online Activation

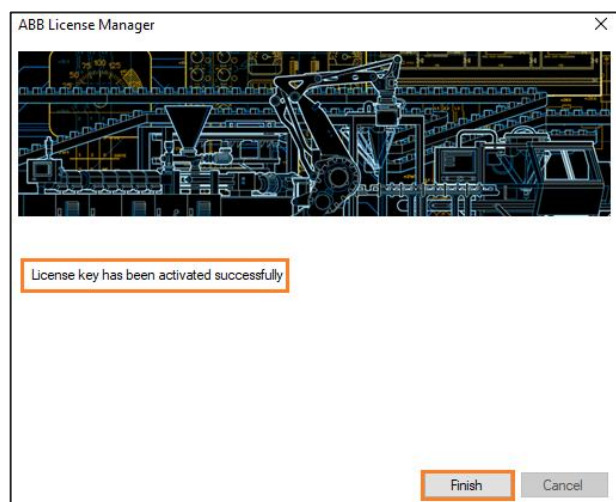
The activation data can be sent to ABB License server if the computer has active internet connection.
Select “Online activation” and click “Next”.



License data now will be transferred to ABB License server.

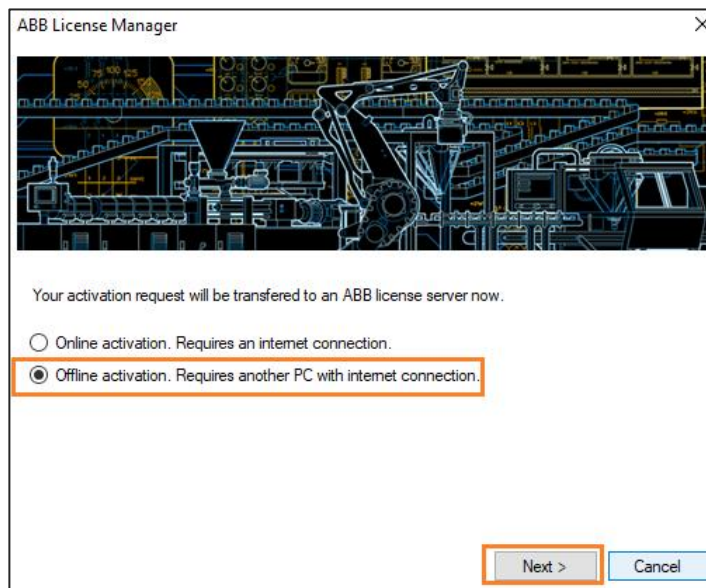


A successfully ended licensing procedure ends with a success message, click “Finish” to end the procedure.



3.2.2 Offline Activation

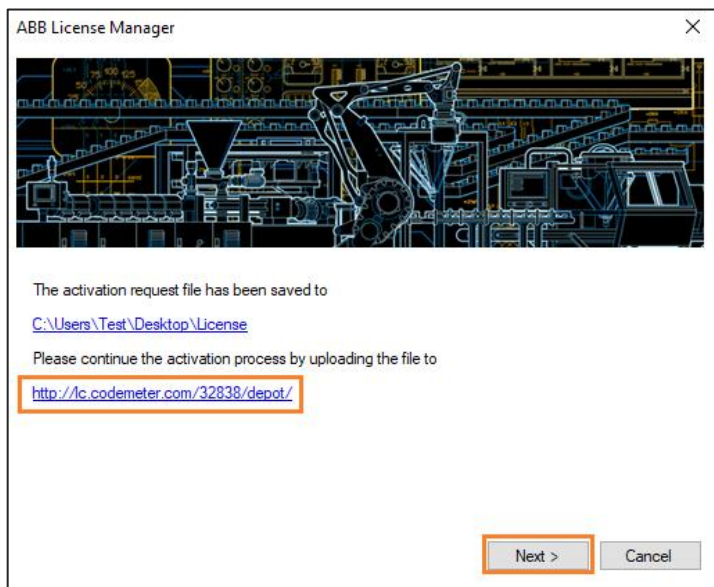
The activation data cannot be sent to the licensing server if the computer does not have internet access. In this case, the offline activation file is created. This file can be used to transfer activation data to the licensing server from another computer with internet access.



Select Offline activation and click “Next”

Enter a file name and click “Save”. An activation file is created and stored to the selected directory.


Transfer the activation file to a computer with internet availability and click “Next”



Upload the offline activation response to the licensing website <http://lc.codemeter.com/32838/depot>

Select the xml file stored previously

Click on “upload request and continue”

HOME English 

Automation Builder - Offline Activation

Upload activation request Download activation response Upload activation receipt

Activating your licenses offline - First step "Upload activation request":


1. Create an activation request file with the Automation Builder Activation Wizard.
2. Pick the created activation request file.
3. Click "Upload request and continue".

Pick activation request file (*.xml)

Choose File AB License.xml

Upload request and continue

Download the activation response.

HOME English 

Download activation response

Upload activation request Download activation response Upload activation receipt

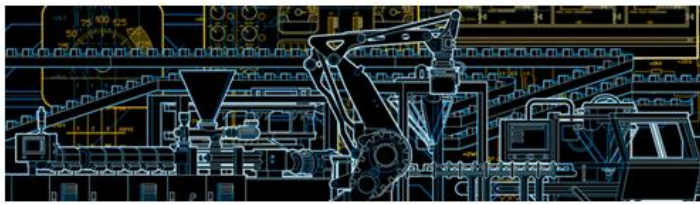
Activating your licenses offline - Second step "Download activation response":

1. Click "Download activation response" and save the file.
2. Import this activation response file using the Automation Builder Activation Wizard.
3. Please confirm the successful activation by uploading the activation receipt file generated by the Automation Builder Activation Wizard afterwards (only supported for Automation Builder 1.1.1 and later).

Download activation response Next

Transfer the activation response file to the offline computer. Select the activation response and click "Next"

ABB License Manager

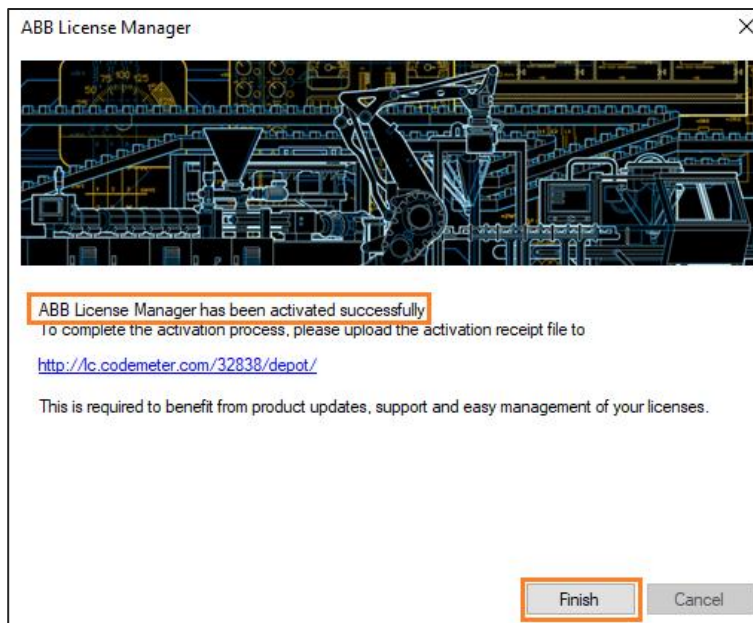


Please provide the activation response file to continue the activation process.

C:\Users\WibuCmRaU Browse

Next > Cancel

A successfully ended licensing procedure ends with a success message, click "Finish" to end the procedure.



Due to security reasons an activation receipt file is created. Save and upload the activation receipt file from the ABB License manager to the registration website <http://lc.codemeter.com/32838/depot/> to complete the license activation process.

3.3 Using Servo Drives with AC500 PLC

When ‘Drives - Servo drives’ is selected in the installation package, the user will get an additional installations of;

- ‘Mint Workbench’. This is the programming and configuration tool for ABB’s current Servo Drives offering, the MicroFlex e190 and MotiFlex e180.
- ‘Mint Sidebar’ which is a tool for easy online access and device management.

The following sections will reference these tools. For additional resources on ABB Servo Drives <https://new.abb.com/drives/low-voltage-ac/servo-products>.

3.3.1 Setting up ABB Servo Drives for use with EtherCAT Master

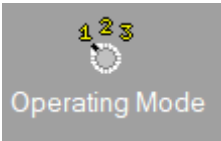
This section assumes that you have already commissioned the drive. i.e., You have been through the Mint Workbench commissioning wizard to define the motor and application settings and have then auto tuned (and fine-tuned if necessary) the control loops for the drive. Details on commissioning the drive can be found in the relevant drive installation manual or you can make reference to application note AN00250 which can be downloaded.

	Value	Mode
HI	00	EtherCAT slave mode
	01-EF	Ethernet POWERLINK CN mode: selected value is node ID
	F0-F1	Reserved
LO	F2	PROFINET slave mode
	F3-FF	Reserved

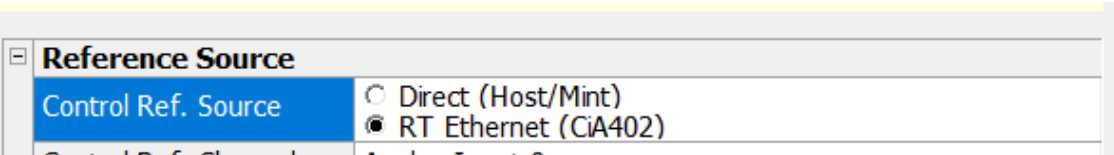
After the drive has been commissioned the user should check that its configured correctly for EtherCAT control. The first step here is to check that the rotary switches are set correctly. As shown below these can define multiple modes of operation and should be set to 00H for EtherCAT Mode

Next we must define the correct ControlRefSourceStartUp, this defines the initial operating mode when the drive is powered up or restarted. The control reference source Start Up can be set using several methods

- the parameter viewer > Configuration Group – Parameter 1.5
- Command Line using CONTROLREFSOURCESTARTUP(0)= crsRT_ETHERNET_402



- Assistant from main view select



Then set ControlRefSource = RT Ethernet (CiA 402) then press the ‘Finish’ Button

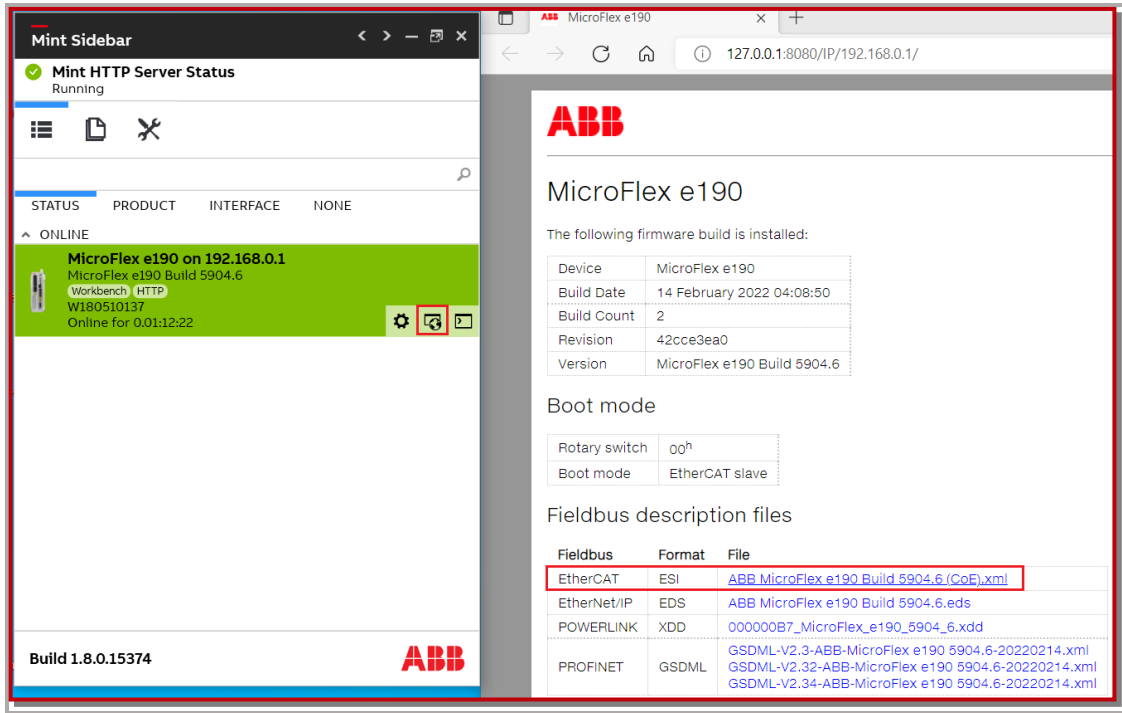
The final step is always to Save the parameters using the save ICON in the tool bar.

Note: Whatever the drives operating mode is set to, when the PLCs EtherCAT master starts it will always force the drive to Real time Ethernet when EtherCAT is started. Despite this selecting ‘Real Time Ethernet’ as the drive’s default source is preferable as this then allows means that no change of control mode is needed.

3.3.2 Exporting the xml file from the drive

Before starting a new PLC, project and going through the navigation of initial configuration we need to make sure that the according .xml file of E190 that we need has been installed into the 'Device Repository', if not, please follow below process:

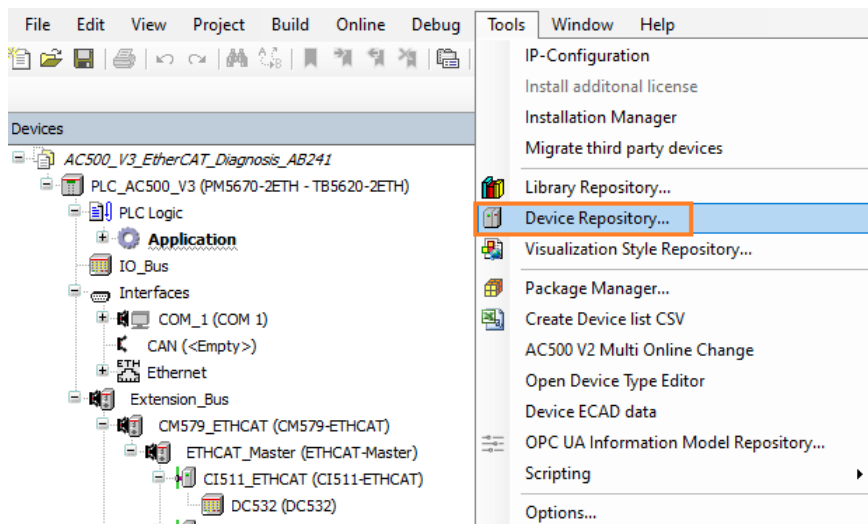
- Open the Mint Sidebar
- Connect to the E190 which will make it appear in the list with a green colour
- then click the 'web' icon. This will open the web server
- Shown at the bottom of the main page a series of files associated with the installed firmware version
- Select the EtherCAT file to initiate the .xml file download to your hard drive:



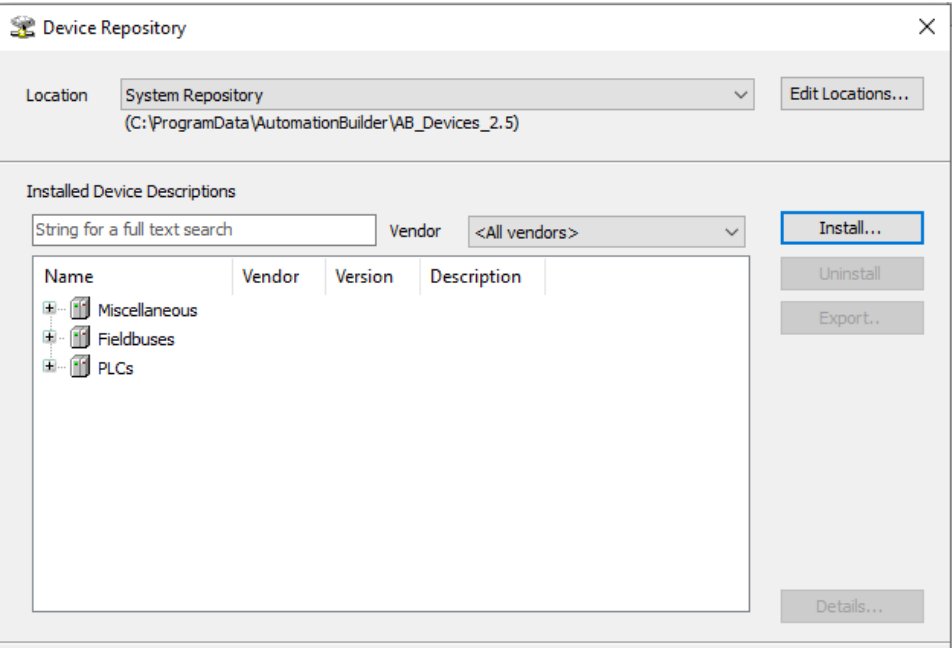
3.3.3 Adding ABB and 3rd party devices to the Device repository

It is recommended to use ABB drives and devices, however the user can also use 3rd party drives and devices by installing the relevant device description files by using Automation Builder. Also this process can be useful if updates are required in between Automation Builder releases.

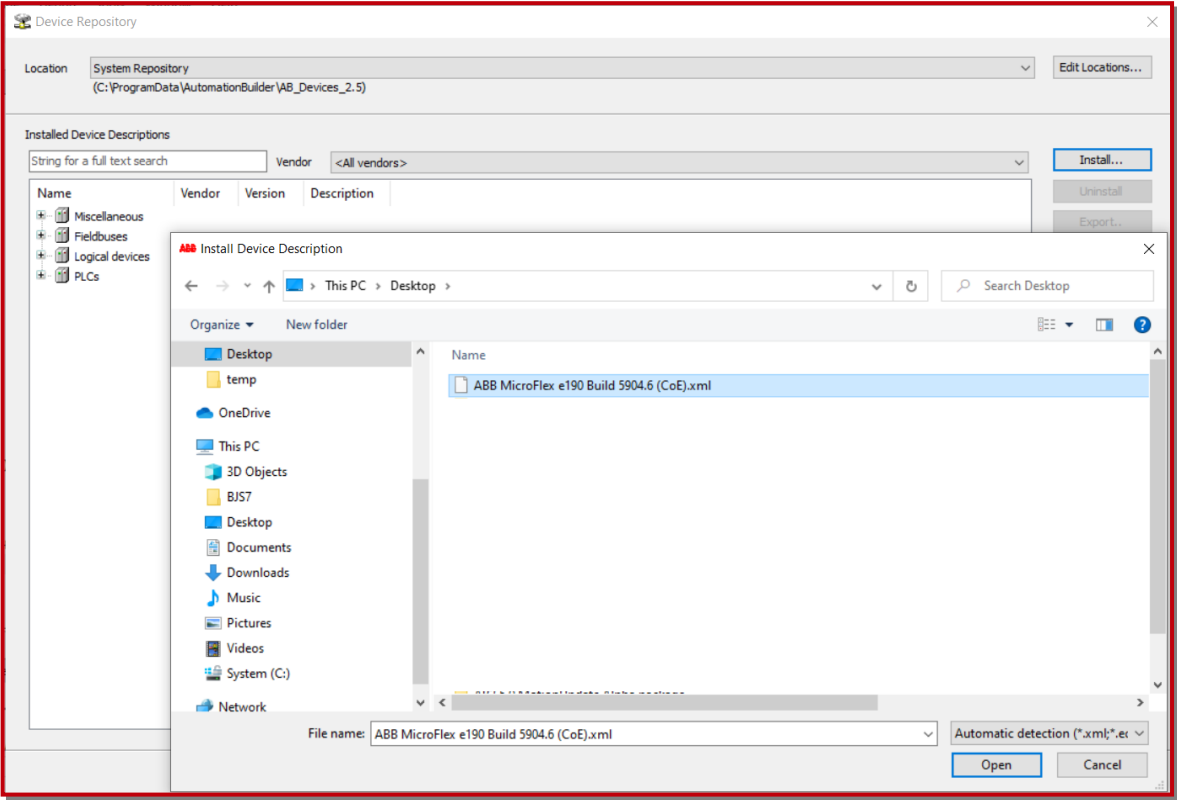
To install the device description files, click on the Device Repository under Tools menu.



This will open the Device Repository window, next click on ‘Install’ and select the device description file which needs to be installed.



Click ‘Install...’ and then find the location where the desired files are stored. Once selected please click ‘Open’.



The status of the file installation will show at the bottom of the window. If successful the device will be immediately on the list of installed devices and the After successful installation, user can now add the installed device under the respective protocol configured in Automation Builder.

After this step is complete, we can use this version into the project as shown next.

4 INTRODUCTION TO THE PROJECT

The following steps show how to set-up a project and configure the hardware. A simple project is used as example to introduce Automation Builder.

The workflow for creation of a visualization is explained, as well as how to set-up a webserver for visualization.



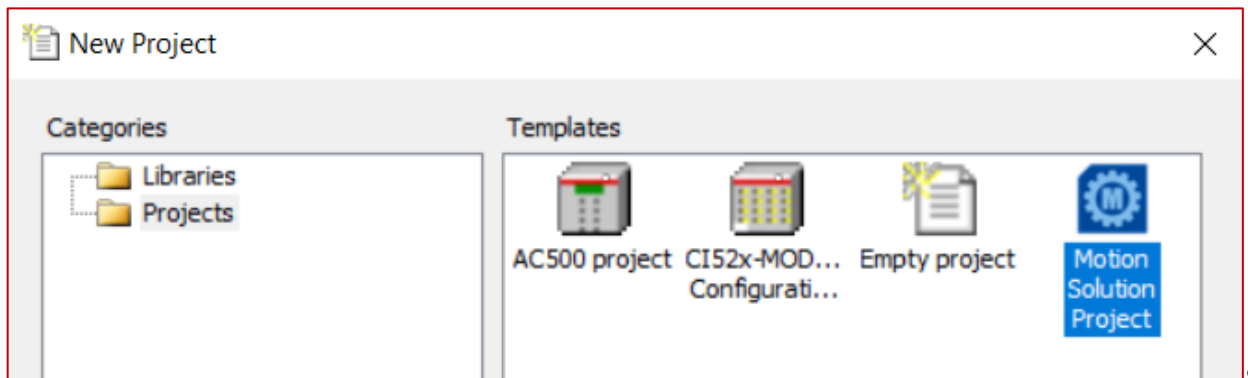
Note: For latest updated information on AC500 products, please always refer to latest Automation Builder online help file.

4.1 Project types guidance

When a new project is started, the correct project type must be selected. Before the correct selection can be done, the project types must be understood.

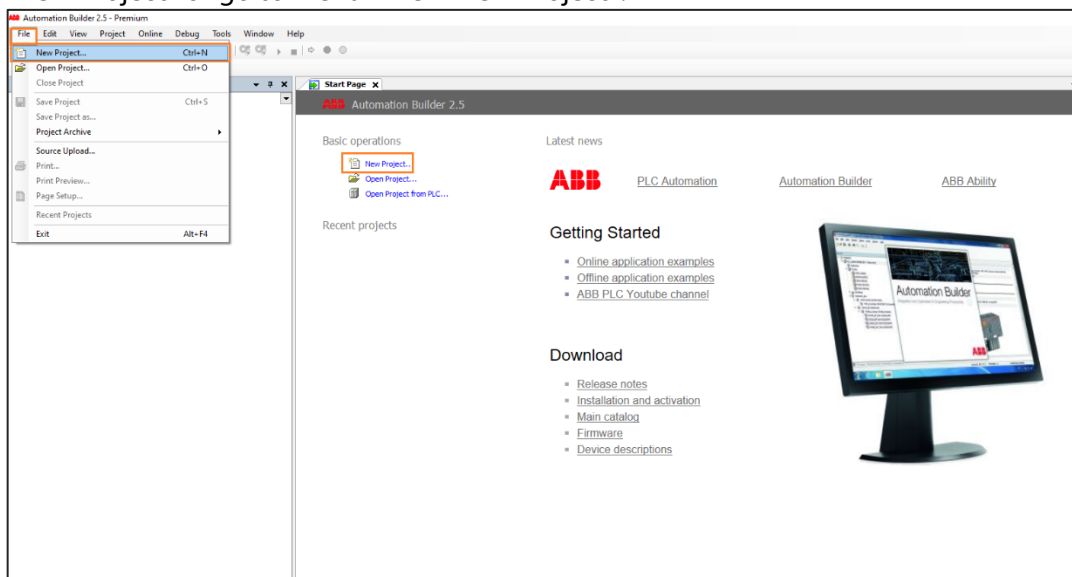
4.1.1 Different project types

To create a new project the user must Launch Automation Builder either out of the desktop icon or out of the Windows menu.



Select

“New Project” or go to menu “File ->New Project”.



Now the user will be prompted to select the Project type. There are different ways of starting an AC500 project as shown below in the New Project window:

- **AC500 Project:** Using this method to build an *AC500 project*, yourself starting from “scratch”. This will take longer to achieve a running program but gives the most flexible in choices.

- **CI52x-MODBUS Configuration** – Not relevant for this Application note
- **Empty Project:** Not advised to use this selection.
- **Motion Solution wizard:** This method will use a guided setup process for configuration of a motion controller project in V3. This will significantly simplify the motion control project start up with PLCopen Motion Control by using (see chapter Motion Solution Wizard)

After the user selects the project type, a project name and location it will be saved to needs to be filled in, then by pressing the 'OK' button the hardware selection can be started.

4.1.2 Understanding when to use the different project types

There are three main motion control methods with AC500 PLCs. These are outlined below along with their characteristics.

Required control Method	Compatible PLC Hardware	Compatible Drive Hardware	Limitation of Movement	Correct project type to select in 'New Project' Window
EtherCAT CiA402 Control	AC500	Any EtherCAT CiA402 Drive	Point to Point Multi Axis Moves	Motion Solution Wizard
24v PTO Control	AC500eCo	Any 24v PTI Drive	Point to Point Multi Axis Moves	AC500 Project Motion Solution Wizard
Modbus GDI	AC500 or AC500eCo	Only ABB Servo Drives with Mint	Point to Point	AC500 Project

Further information

For further information on the EtherCAT CiA 402 Method see section **Add PTO axis** of this manual for further guidance

For further information on the 24v PTO Control Method see section **Add EtherCAT axis** of this manual for Further guidance

For further information on the Modbus GDI got to <https://new.abb.com/drives/low-voltage-ac/servo-products> and download Application Note: GDI AN00501 - Generic Drive Interface - AC500 V3 Modbus

4.2 Selecting hardware used in the project

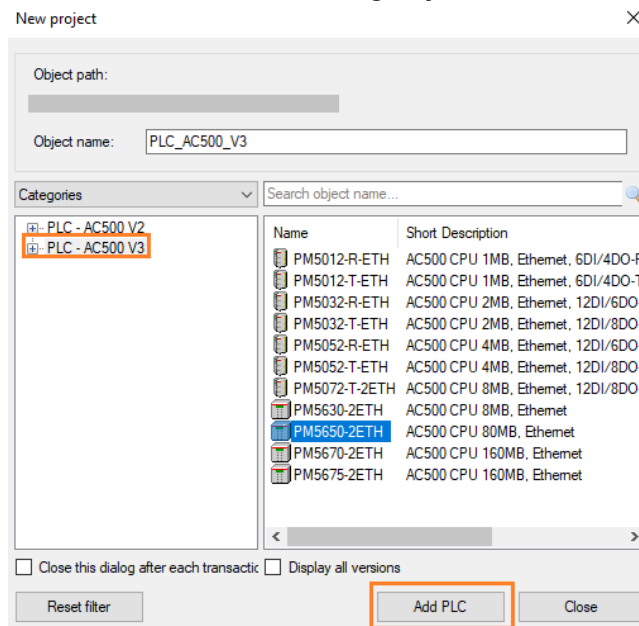
This subchapter describes in short for completeness and reference the general use of AC500 platform.

If you have limited time and want to start directly with Motion Control and AC500 Motion Controller Kits: It is recommended to jump directly to the chapter 8 for the Motion Solution Wizard.

4.2.1 Select PLC Type

1. To see a list of only AC500 V3 CPU's first select "PLC - AC500 V3" in the categories window

2. Select the CPU according to your hardware set-up.

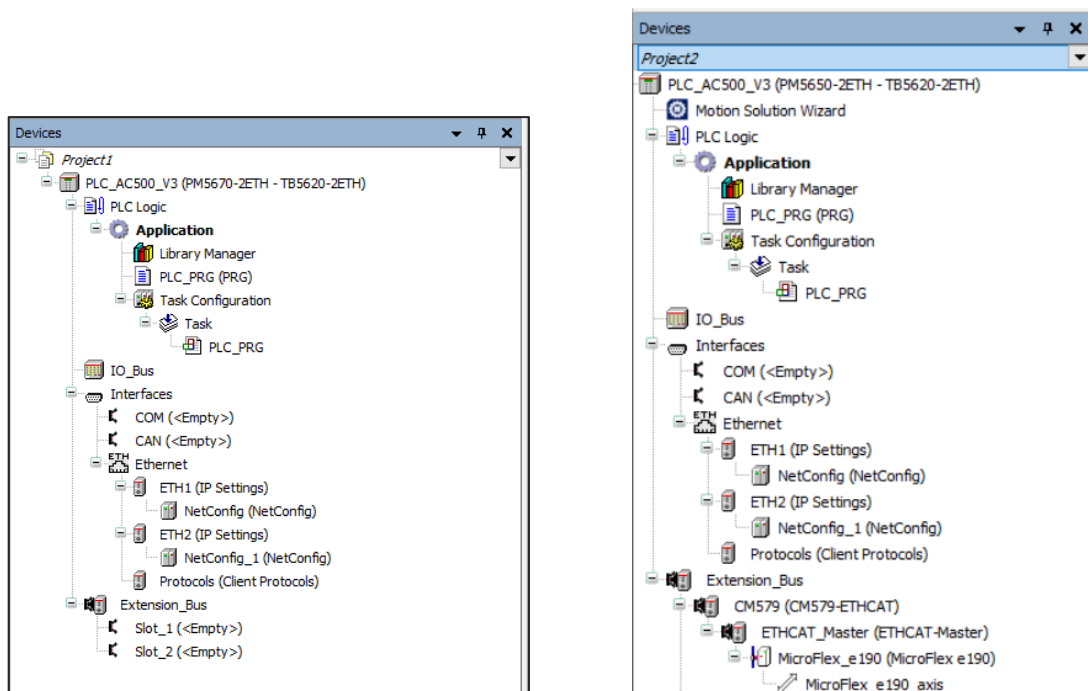


3. Select “Add PLC” to add the CPU to your application.

4. Next


- you will either be prompted to select further hardware options if your project type uses the Motion Solution Wizard (for Motion Solution see chapter 8)
- or if not will build the Devices tree for the project with the default settings.

Below on the left we can see the hardware tree from an “AC500 project” and on the right, we can see the same from a “Motion Solution” project (for Motion Solution project see chapter 8):



4.2.2 Saving the project

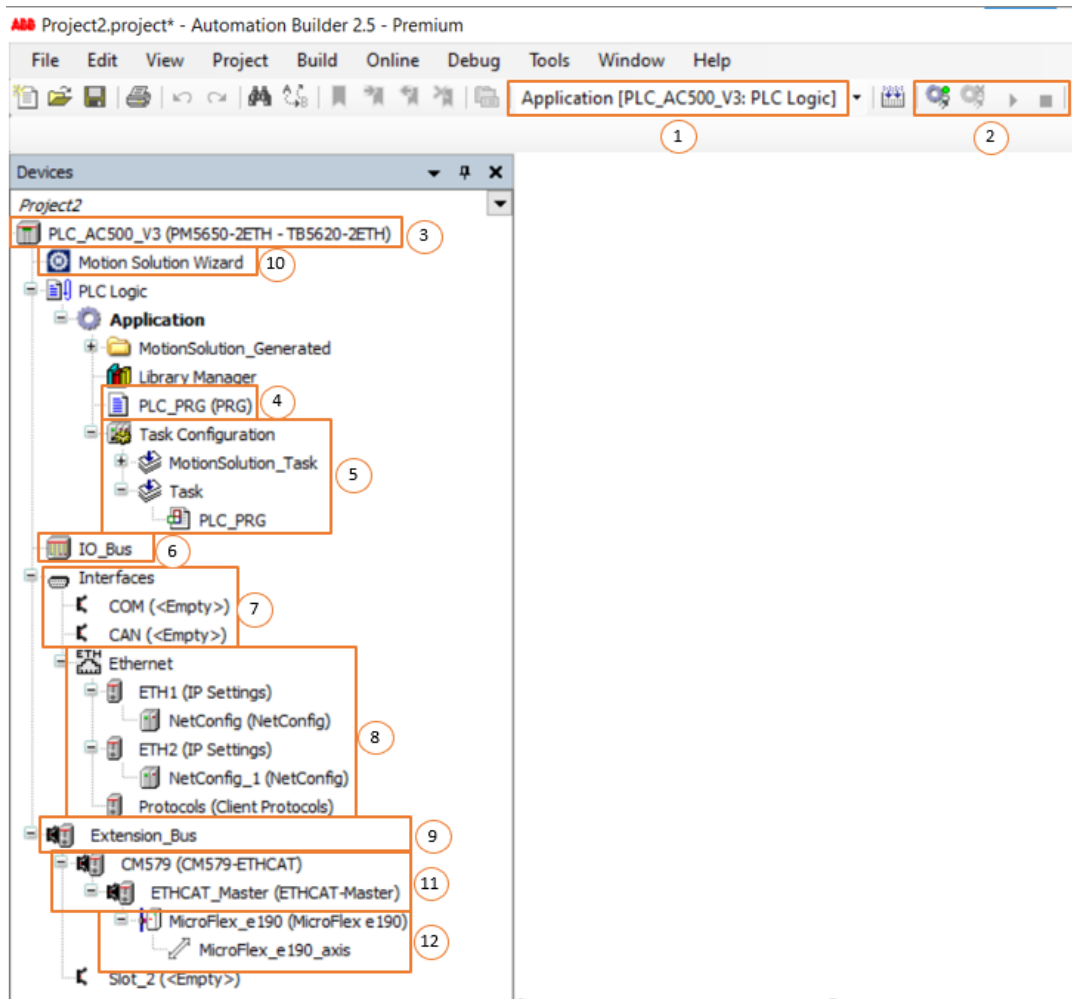
User can save the created project any time with below options,

- Select menu “File -> Save Project”.
- Alternatively, select the save icon  in the tool bar.

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

4.2.3 Navigating the project

Below highlighted is the Automation Builder default layout, user can add additional objects which will change the layout accordingly.



Items shown above

1	Select the active application	2	Login & Logout / Run & Stop
3	PLC Type	4	IEC program
5	Task configuration	6	Local IO bus
7	Serial interface	8	Onboard Ethernet
9	Extension bus	10	Motion Solution Wizard
<u>Motion Project Specific items:</u>		11	EtherCAT Master Coupler
		12	EtherCAT Slave Devices

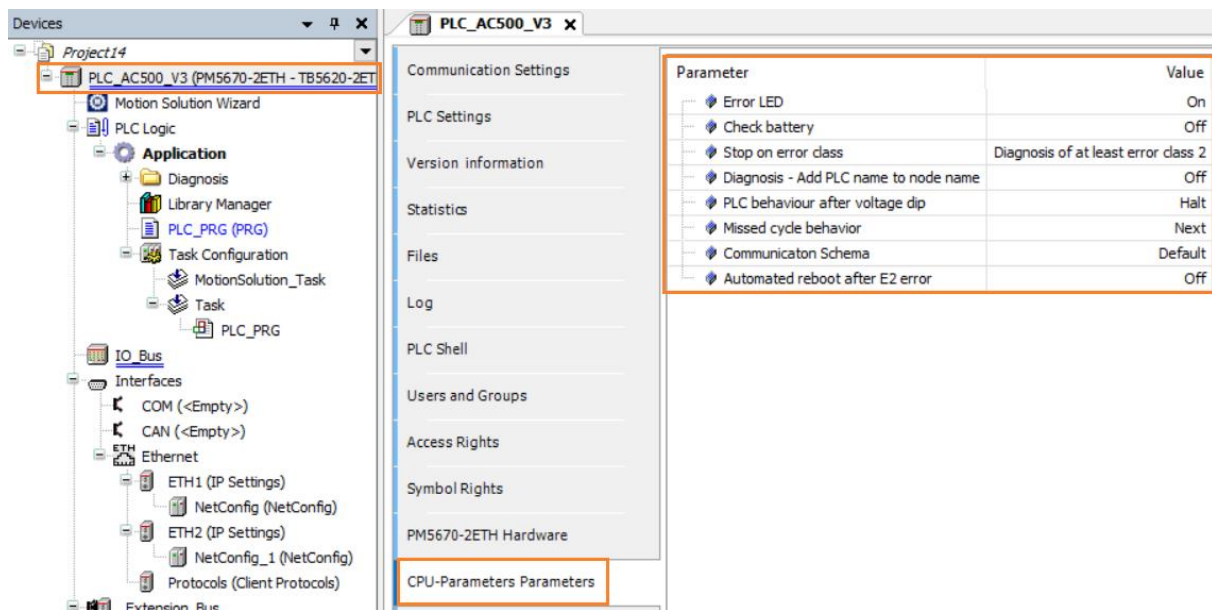
4.3 Important CPU parameters

CPU parameter settings should be considered when configuring the Automation Builder projects at least for demanding cases where defaults are not fitting. Please check and update the parameter as per your hardware setup and system requirement. To access these settings.

Double-click “PLC_AC500_V3”.

A tab opens in the editor view.

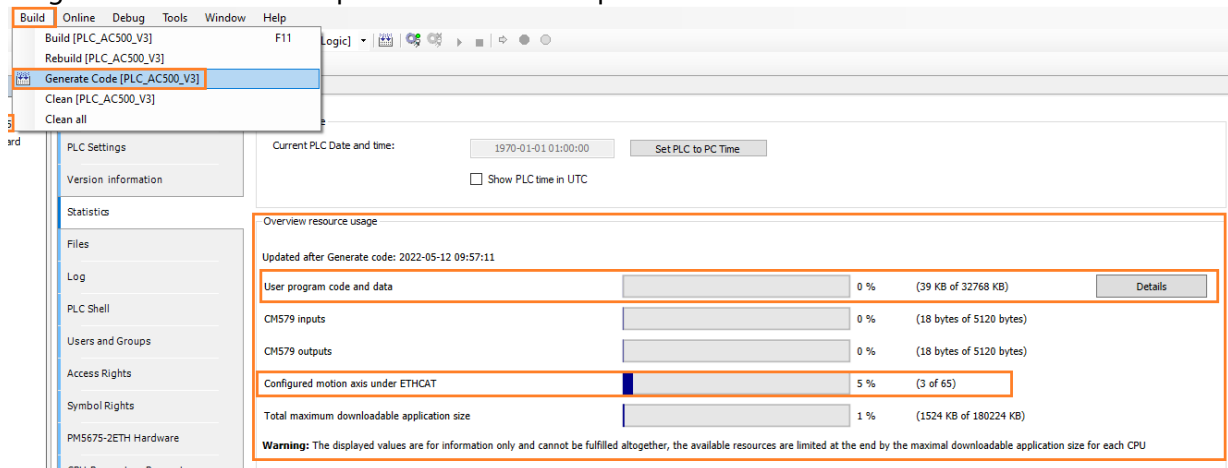
Select “CPU-Parameters Parameters”.



For example, if your CPU does not have a battery installed, then you can go to the parameter “Check battery”, choose the value “Off” to avoid the error message in the PLC after downloading the program.

4.3.1 Checking program size and number of configured axis

User can check the program size and the number of configured and supported axis from Statistics tab. To get the Statistics tab updated user need to perform “Generate Code” from “Build” menu.



Overview resource usage:

This tab shows all the required information (it is collected at latest when the command “Generate Code” is executed, some of the information is not available before then.)

For the limits of “User program code and data” a [Details] button is available. Clicking this button will open a window showing a more detailed view of the memory usage.

This tab also shows the configured and maximum supported motion axis. The supported motion axis is limited based on the PLC type and EtherCAT cycle time (Refer the chapter “Limits on number of synchronized axis” for more details) but has a limit higher by one axis - to account for at least one of the typically needed virtual axis.

4.4 Changing CPU type

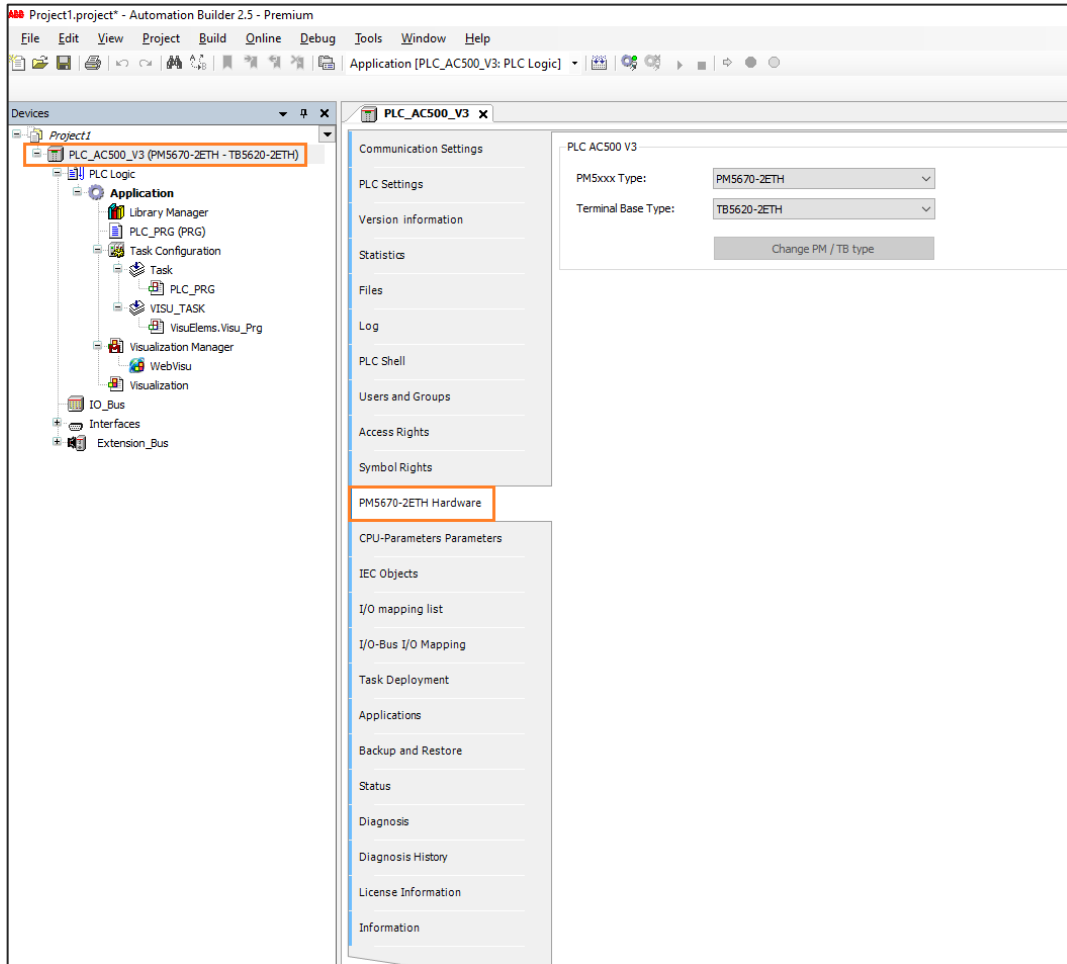
User can easily switch a project from one PLC / terminal base to another PLC / terminal base using the using the change PM/TB option in Automation Builder.



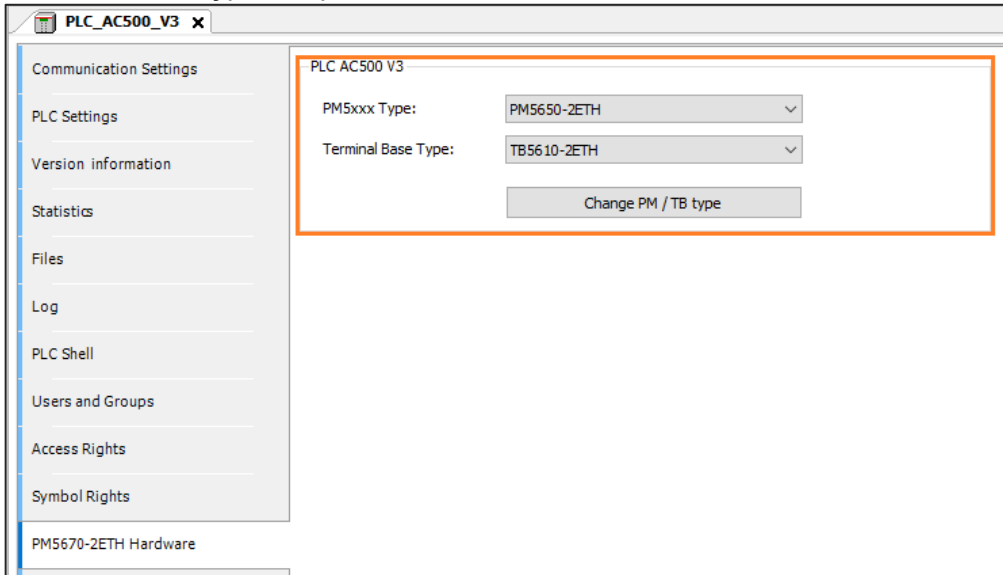
Note: When the PLC type is changed with in the same series PLC (V3 to V3), it is expected to work without any additional effort however depending on the new PLC type, user may need to check the user program / cycle time to make sure the performance is optimal with new PLC.

Follow the below steps to change the PLC type,

Double-click the PLC_AC500_V3 <...> node and open the “PM5<...> Hardware” tab.



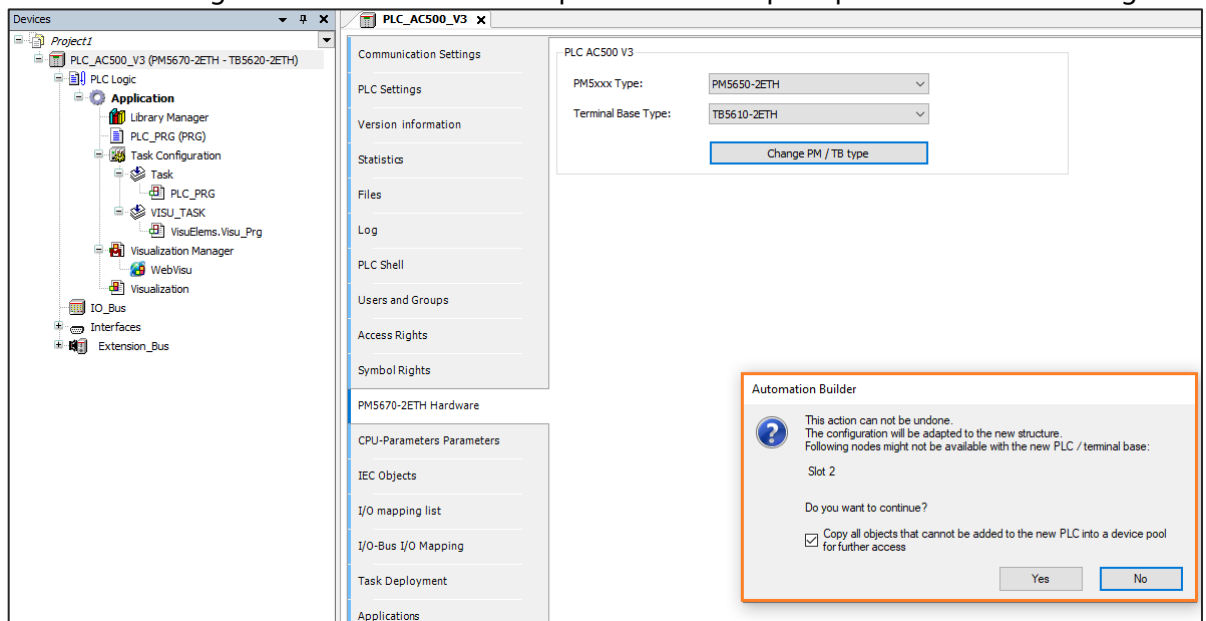
Select the desired V3 PLC from the “PM5xx Type” drop-down list and the correct terminal based on the “Terminal Base Type” drop down list.



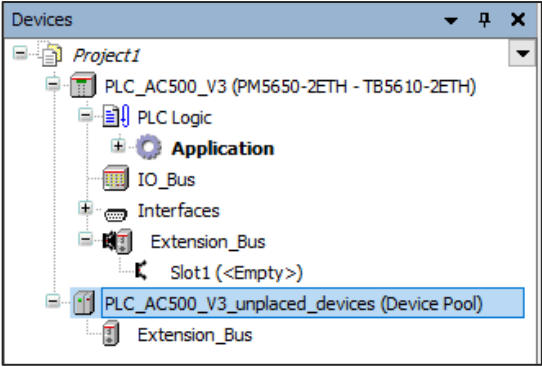
After selecting both PLC type and terminal base, click on the “Change PM/TB type”.

If possible, the device configurations from the previous processor module will be kept and switched over to the new processor module.

The device configurations that cannot be kept are listed in a prompted information dialog.



By default, all device configurations which cannot be switched over will be copied to a "device pool" section in the navigation tree (option “Copy all objects that cannot be added to the new PLC into a device pool for further access”). If required, this backup configuration can be used in another project or in another processor module configuration.



If the checkbox is deactivated all device configurations that cannot be switched will be lost after the execution of the target change.

4.5 I/O in AC500 and S500 IO System

For AC500 local or remote I/O can be added out of the S500 I/O System offering (+ integrated I/O are available in the AC500 eCo CPUs).

Local I/O: up to 10 S500 I/O modules can be added on the right side of AC500 and AC500 eCo via the TU terminal units.

Remote I/O: the S500 CI communication interfaces (which already have integrated I/O channels!) can add again up to 10 more I/O modules each.

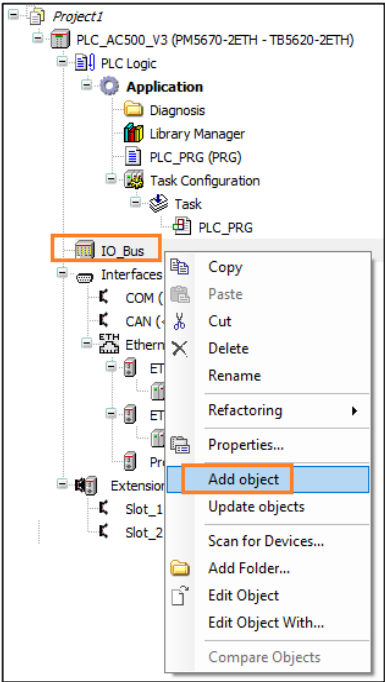
directly via CPU integrated protocols such as Modbus, CAN

or via a chosen field-bus and the matching CM communication modules (left side of AC500 CPU) acting as a bus coupler.

4.5.1 Configuring local ABB I/O module (S500)

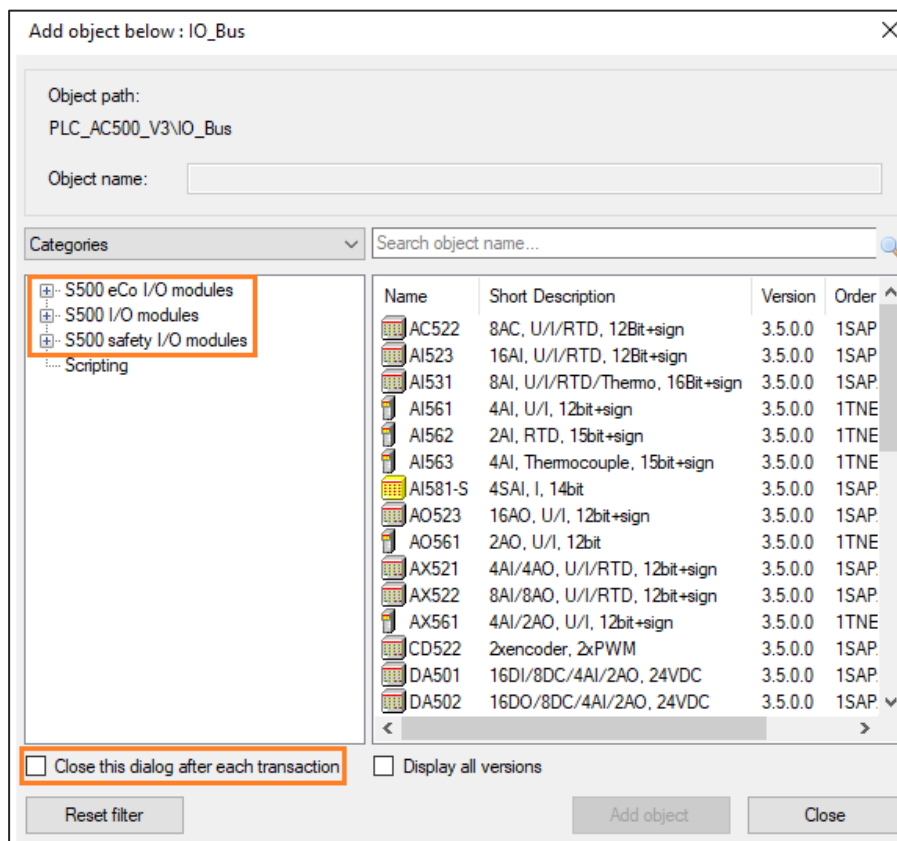
The types and order of modules in the Automation Builder project must match the real hardware configuration. The position of the modules in the device tree can be changed by drag and drop.

- Right-click “IO_Bus” in the device tree.
- Select “Add object”.



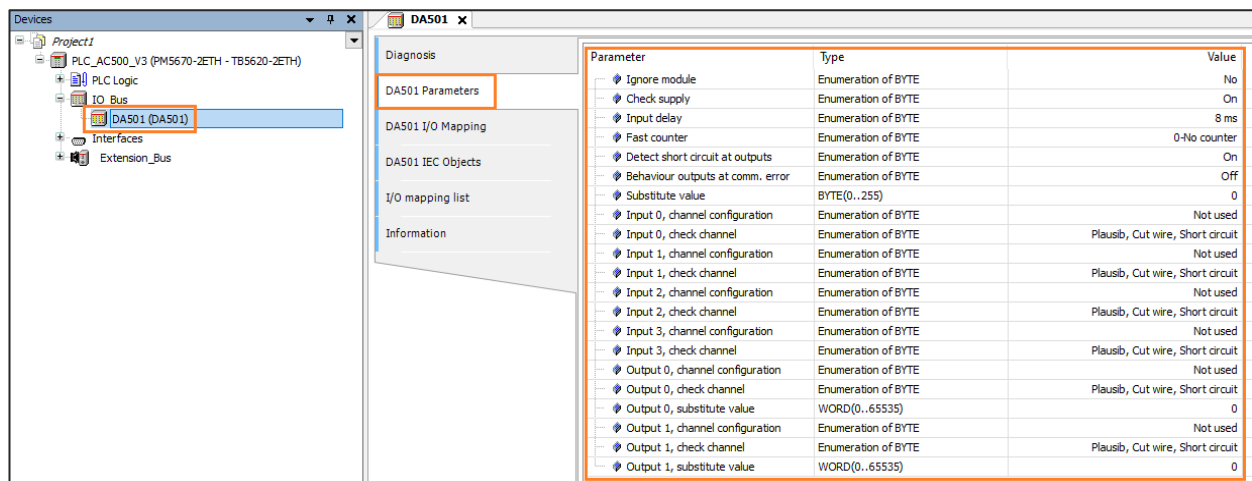
1. Select S500 modules as per the real hardware available and in the same order how they are physically connected.

Uncheck the box “Close the dialog after each transaction” if you have multiple modules to be added.

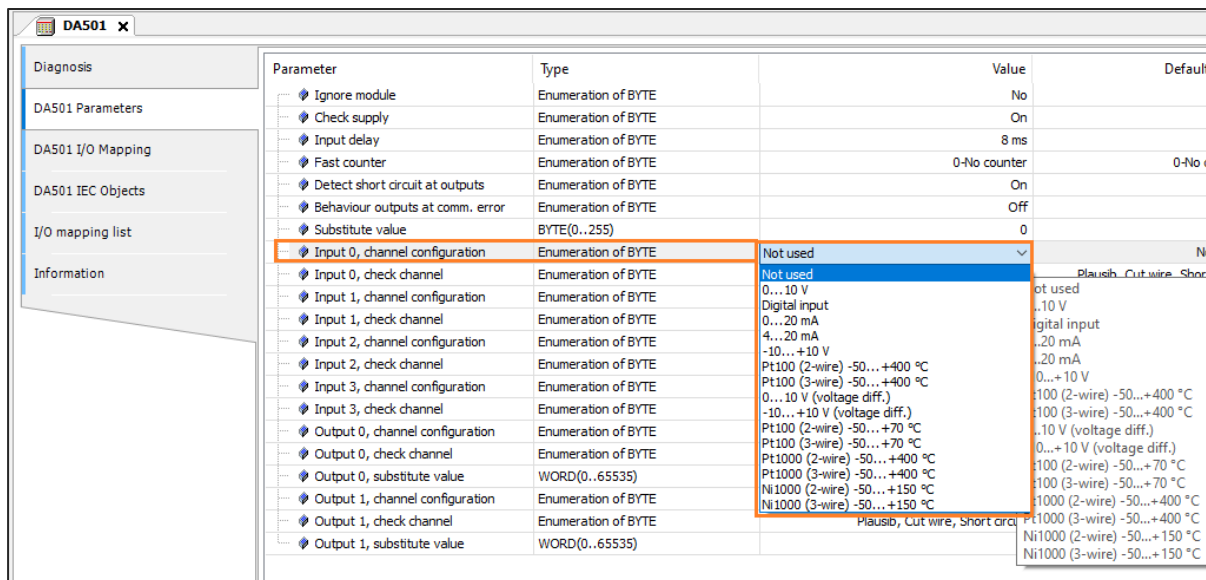


2. Select the module and click on “Add object” to add the module to the I/O bus.
3. After adding the IO modules, double click on the added IO module which will then open a tab in the editor view, click on the Parameter tab for configuring the module parameter.

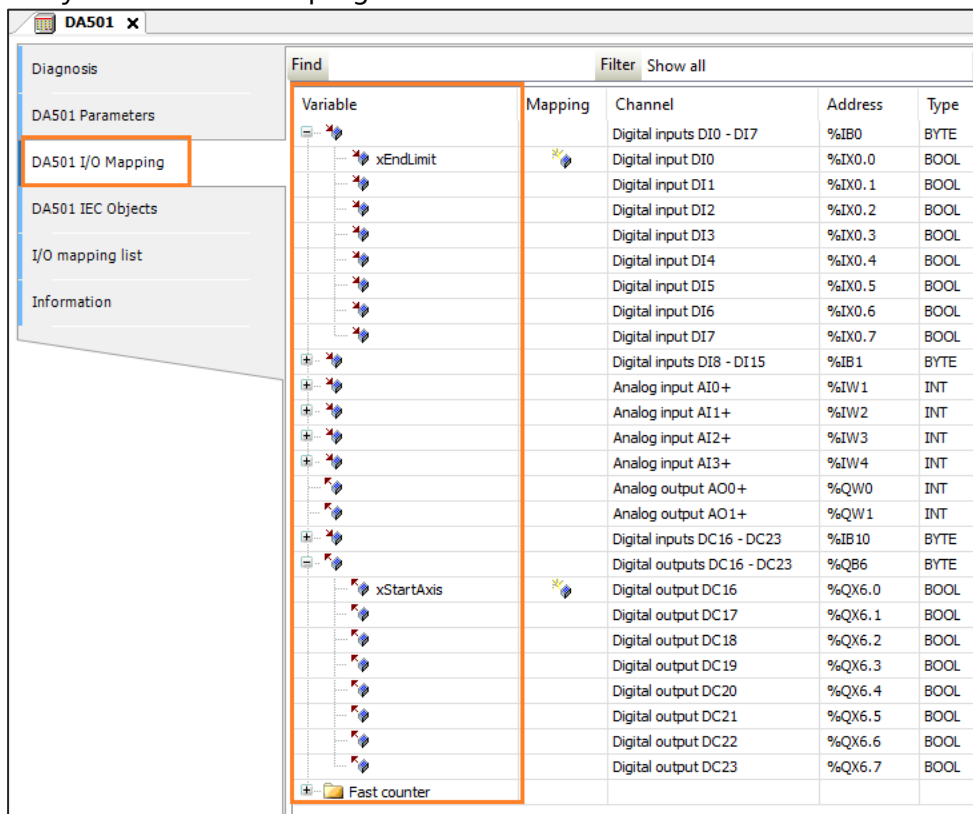
This view will be different for each module based on the module type and needs to be configured accordingly.



For example, by default all the analog channels will be configured as “Not used” and to be configured as per the project requirement to use the analog channel.

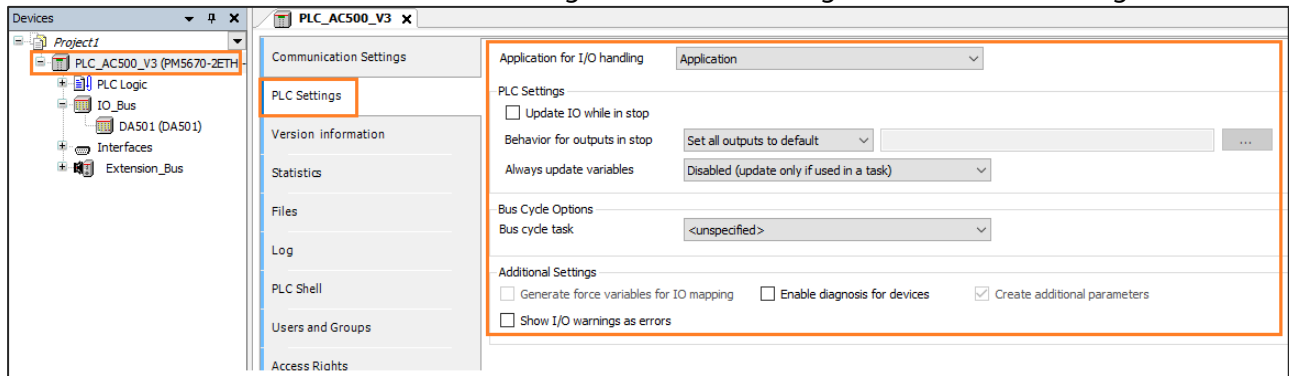


- To add the IO mapping, double click on the added IO module which will then open a tab in the editor view, click on the IO mapping tab for adding the IO mapping. Here you can add the variable names for the channels you will need in the program.

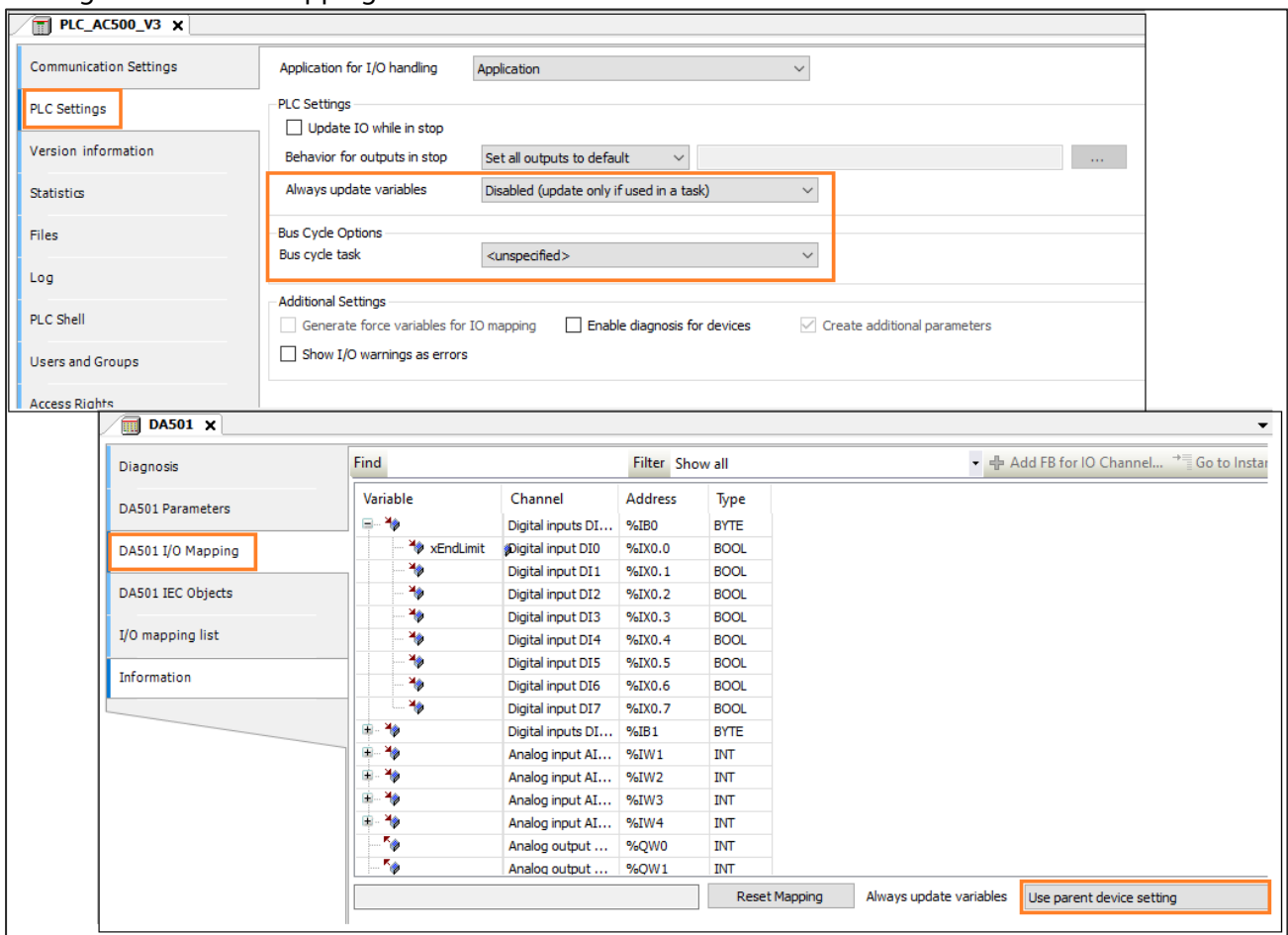


The suggested name convention is based on "Hungarian notation". A name prefix is describing variable type: e.g., "x" = variable of type BOOL, "w" = WORD, "i" = INT (integer) etc. This increases the code readability and is helpful for program analysis.

5. There are some additional IO modules setting which can be configured from “PLC Setting” tab.



For example, by default, in AC500 V3 PLC, if the assigned tag in the IO mapping is not used in the program, tags will not be updated. If the IO mapping to be updated always, user can change the setting in the “PLC Setting” or in the “IO Mapping”.



4.5.2 Configuring to ABB Remote IO

To configure ABB remote IO's user need to configure the protocol in Automation Builder device tree.

User can either use CPU integrated protocols such as Modbus, CAN or via a chosen fieldbus and the matching CM communication modules (left side of AC500 CPU) acting as a bus coupler.

Below is a sample configuration for EtherCAT based IO module.

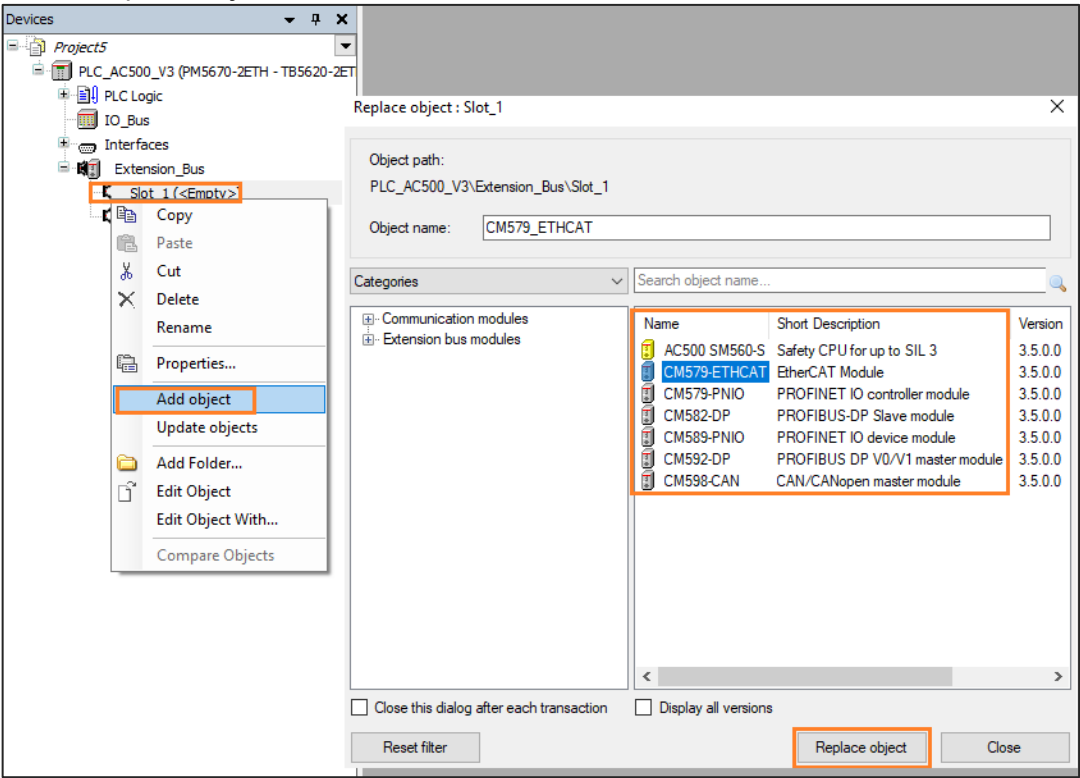
To add the communication couplers on Automation Builder, follow the below steps,

In the Automation Builder device tree under “Extension_Bus”, right-click “Slot_1”.

Select “Add object”.

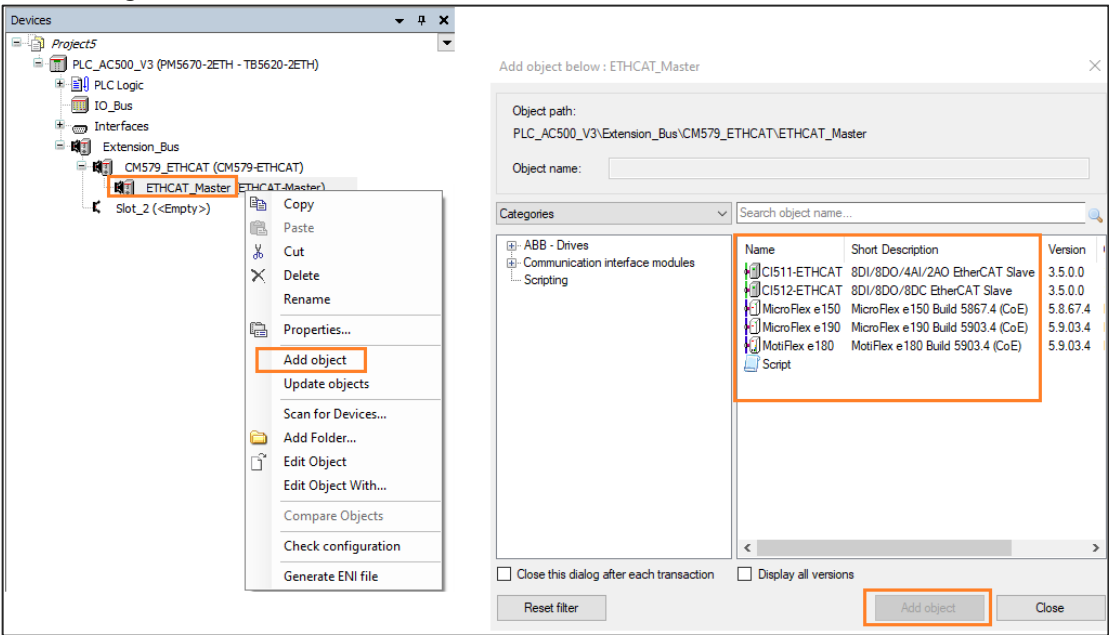
Select the communication module required by the user

Select “Replace object” to add the selected module to Automation Builder device tree.

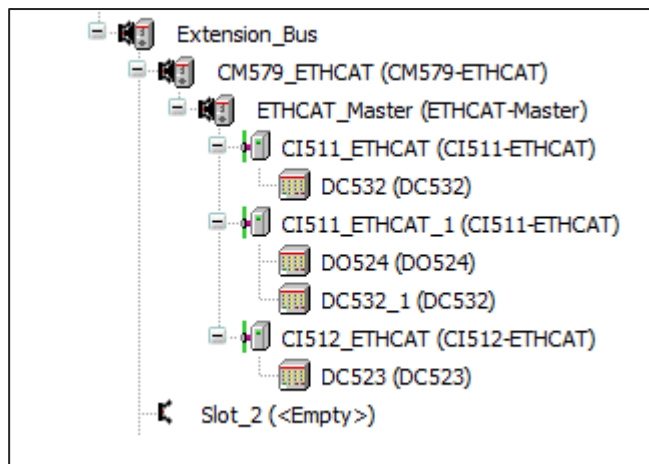


After adding the communication module, user can now configure the added protocol by double clicking on the same which will then open a separate tab in the editor interface and user can configure.

Based on the protocol added, user can add supported devices below and can configure the same by double clicking on the same.



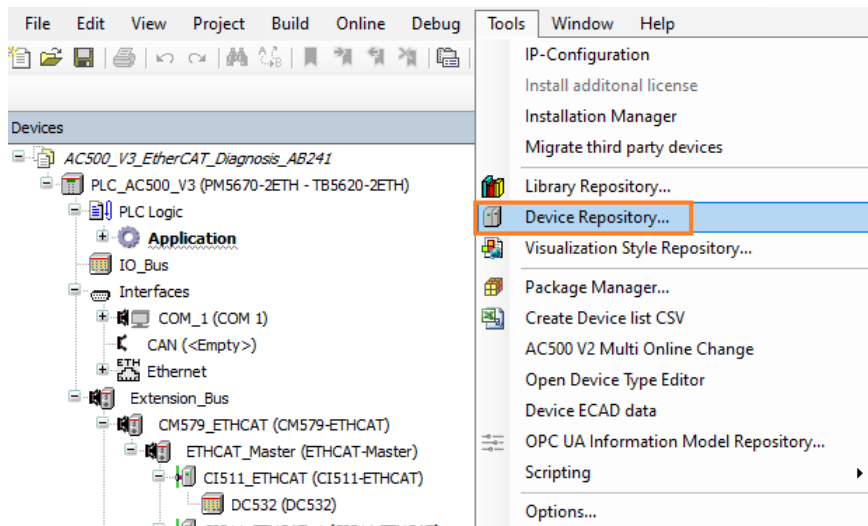
For a properly working EtherCAT system, the Topology of the configuration must match to the one of the setup.



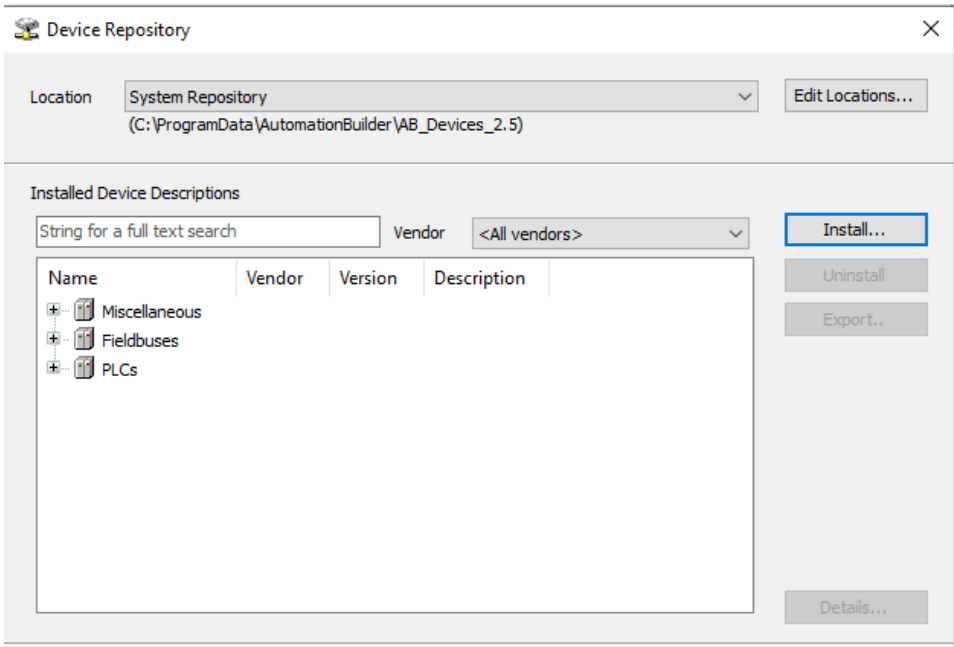
4.5.3 Configuring to 3rd Party Remote IO

It is recommended to use ABB remote IO's, however user can also use 3rd party remote IO's by installing the relevant device descriptions using Automation Builder.

To install the device descriptions, click on the Device Repository under Tools menu.



This will launch the Device Repository, click on install and select the 3rd part device description file which needs to be installed. Successful installation of the file will show the device immediately on the list of installed devices.



After successful installation, user can now add the installed device under the respective protocol configured in Automation Builder.

4.6 Fieldbus protocol types

AC500 PLC supports many communication protocols via onboard interface or additional coupler modules. Based on the application requirement, user can select the respective protocols and configure it using Automation Builder.



Note: For details on supported protocols license requirements, refer latest AC500 catalog or Automation Builder online help file.

4.6.1 Communication using Onboard Ethernet Ports

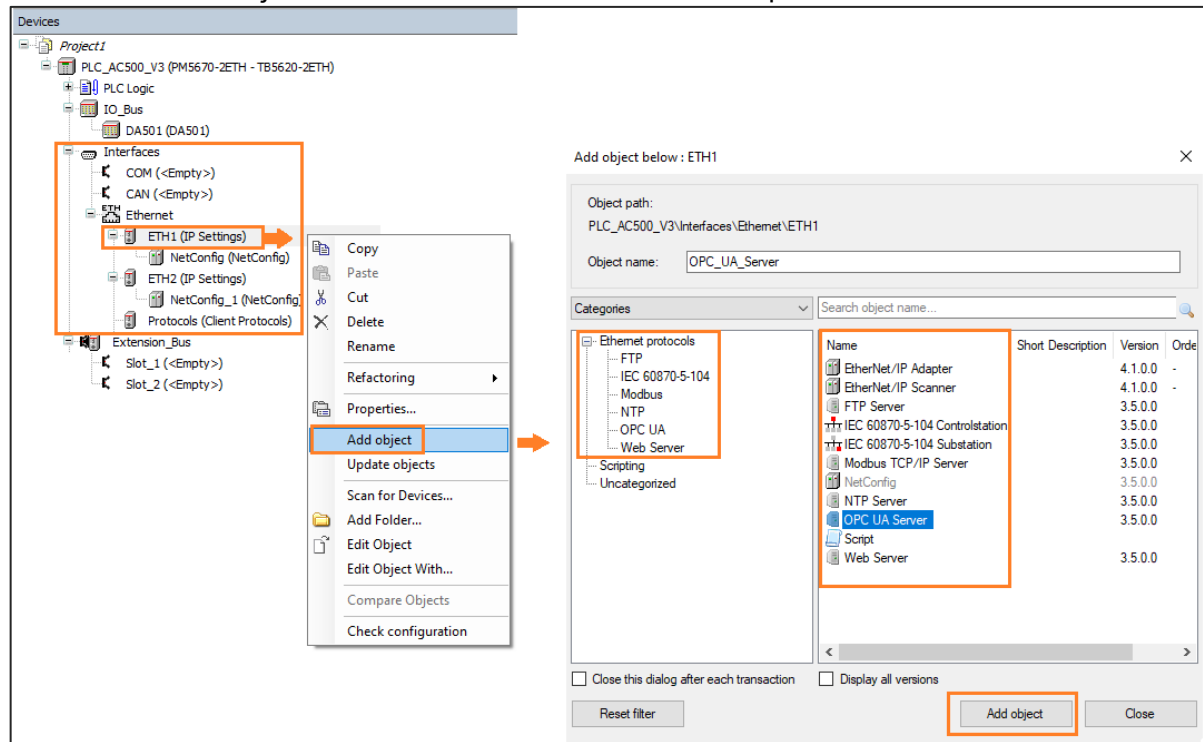
AC500 V3 PLC are having two Ethernet interface and one serial interface which are all configurable for different protocols and one dedicated CAN interface onboard. By default, these ports are not configured, and user need to configure them using Automation Builder. To configure the interface,

Right click on the respective interface on the Automation Builder device tree.

Click on “Add object”. This will then open a separate window with supported protocols on the selected interface.

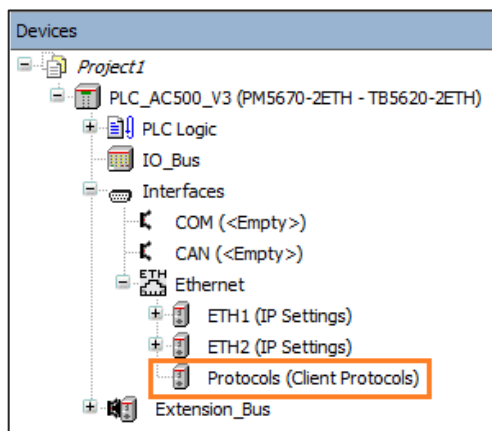
Select the protocol which needs to be configured.

Click on the “Add object” on the window to add the selected protocol on the interface.



Some protocols may need additional configuration and programming after adding the protocol on to the interface, double click on the added protocol, which will then open a separate tab in the editor interface for additional configuration.

To add Client protocols under ETH ports, right click on the “Protocols (Client protocols)” object and follow the same above steps. Client protocols are applicable for both ETH ports.



Note: For more details on configuration and programming needed for the respective protocol, please refer Automation builder online help or [ABB library](#)

4.6.2 Communication via a coupler Communications module

Communication modules (CM) act as couplers or coprocessors and can be added on the left side in slots of the CPU’s terminal base. These CM modules then provide additional interfaces and protocols which can be configured dependent on the CM module chosen. Based on the CPU type and terminal base selected, user can add up to six CM modules.

If a Motion Solution Project is selected, then an EtherCAT communications module will be automatically added to the project in Slot 1. The below section will explain how to add modules manually.



Note: For details on supported protocols license requirements, refer latest AC500 catalog or Automation Builder online help file.

4.6.2.1 Manually adding a communications module

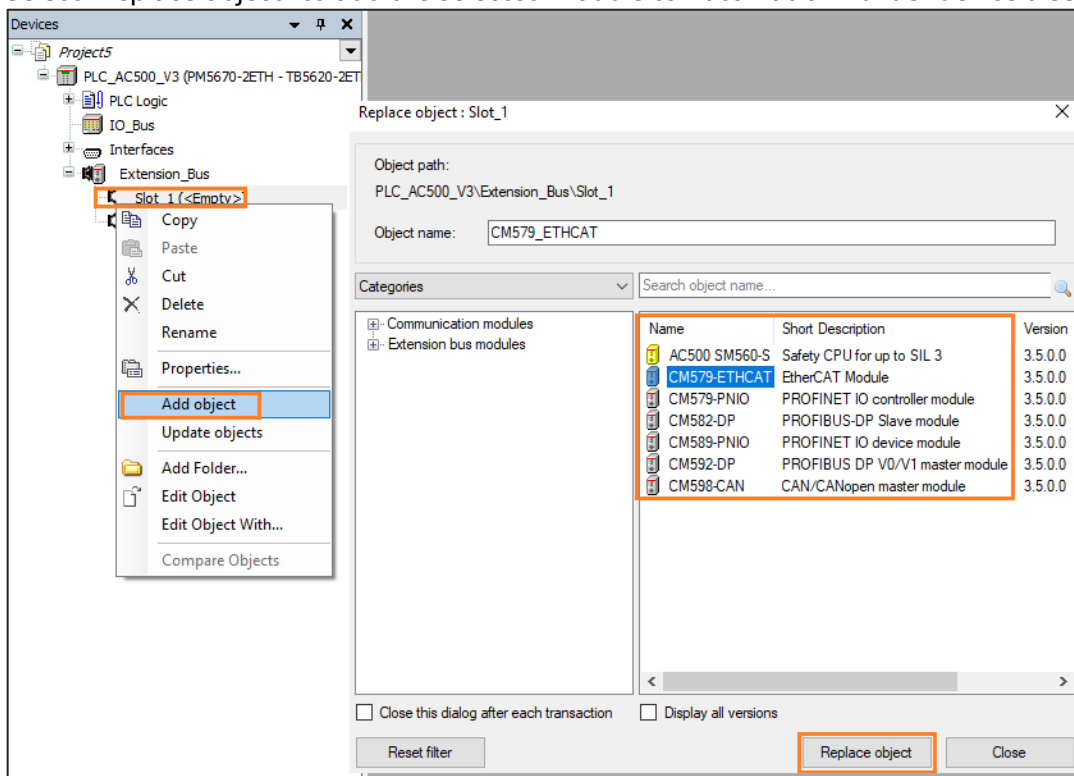
To add a communications couplers to the Automation Builder project, follow the below steps,

In the Automation Builder device tree under “Extension_Bus”, right-click “Slot_1”.

Select “Add object”.

Select the communication module required by the user

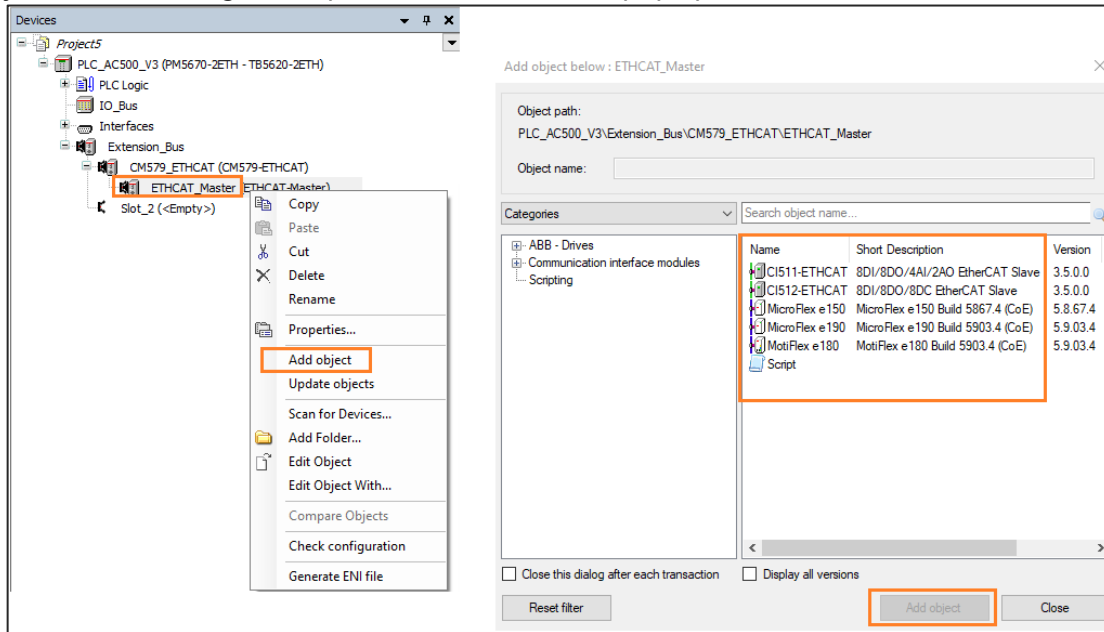
Select “Replace object” to add the selected module to Automation Builder device tree.



After adding the communication module, user can now configure the added protocol by double clicking on the same which will then open a separate tab in the editor interface and user can configure.

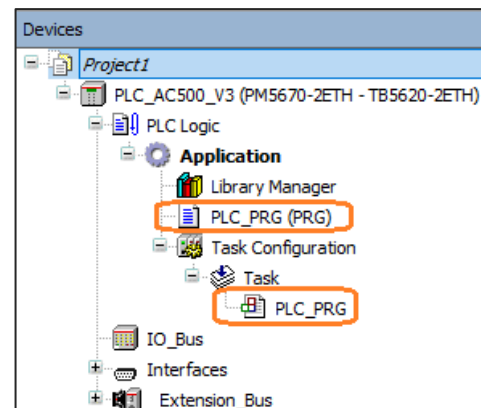
4.6.2.2 Manually adding a communications Slave node

Based on the protocol added, user can add supported devices below by right clicking, selecting “add object” and selecting the required devices from the pop up list..



4.7 Programming and compiling AC500 code

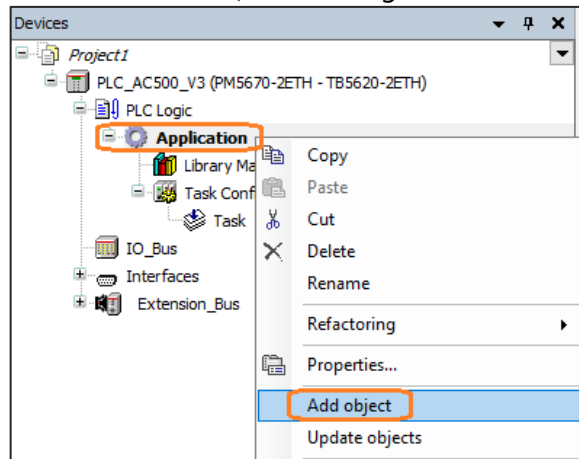
In the default device tree configuration, there is one call of a POU (program organization unit) i.e., "PLC_PRG". User can keep the default POU which is created with structure text language or remove the same and create own POU's in any one of the supported languages.



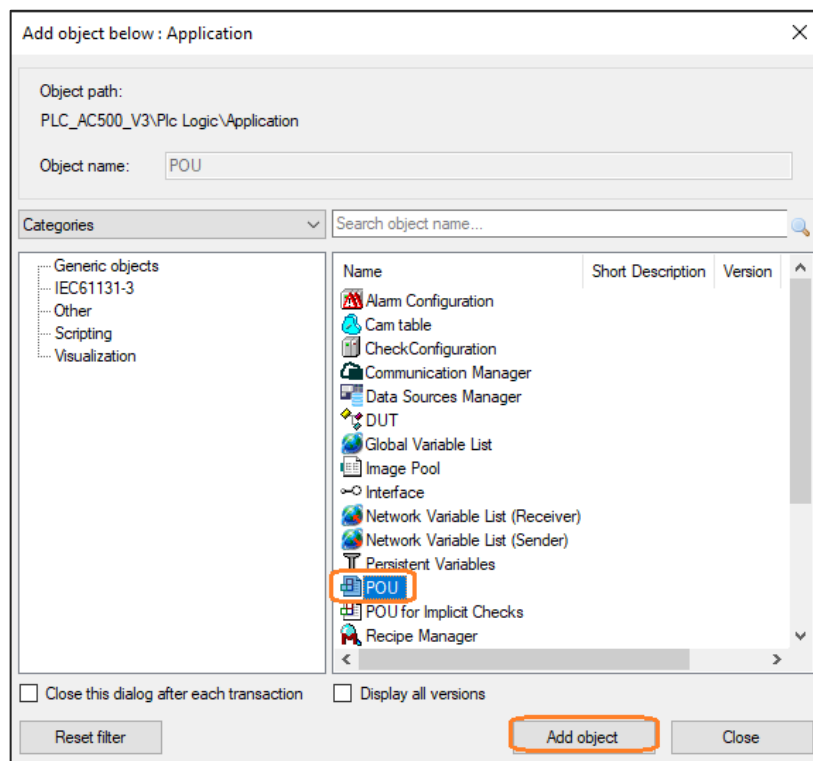
To remove the POU, simply click on the POU and delete.

Please note, this will only delete the POU from device tree, to remove the POU call from task configuration, select the POU name under the “Task configuration” and delete. If the POU is called in other places in the program, user need to manually remove all of them.

To add a new POU, user can right click on the “Application” and select “Add object”



Add object window will be open with all supported objects and user can select POU and click on “Add object”.



This will open “Add POU” window and user can provide POU a name, type and language and click on

Add POU

Create a new POU (Program Organization Unit)

Name
POU

Type
☒ **Program**
☐ **Function block**
 ☐ Extends
 ☐ Implements
 ☐ Final ☐ Abstract
 Access specifier
 Method implementation language
 Continuous Function Chart (CFC)
☐ **Function**
 Return type

Implementation language
 Function Block Diagram (FBD)
 Continuous Function Chart (CFC)
 Continuous Function Chart (CFC) - page-oriented
 Function Block Diagram (FBD)
 Ladder Logic Diagram (LD)
 Sequential Function Chart (SFC)
 Structured Text (ST)

“Add” button to add the new POU on the device tree.

User can add as many as POU’s needed in different languages by following the above steps and add extensions of the IEC 61131-3 standard by right clicking on the POU and “Add Object” and write the logic here.

Devices

Project1

PLC_AC500_V3 (PM5670-2ETH - TB5620-2ETH)

PLC Logic

Application

Library Manager

POU (PRG)

Task Config

Task

IO_Bus

Interfaces

Extension_Bus

Copy

Paste

Cut

Delete

Rename

Browse

Refactoring

Properties...

Add object

Update objects

Add object below : POU

Object path:
PLC_AC500_V3\Plc Logic\Application\POU

Object name:

Categories

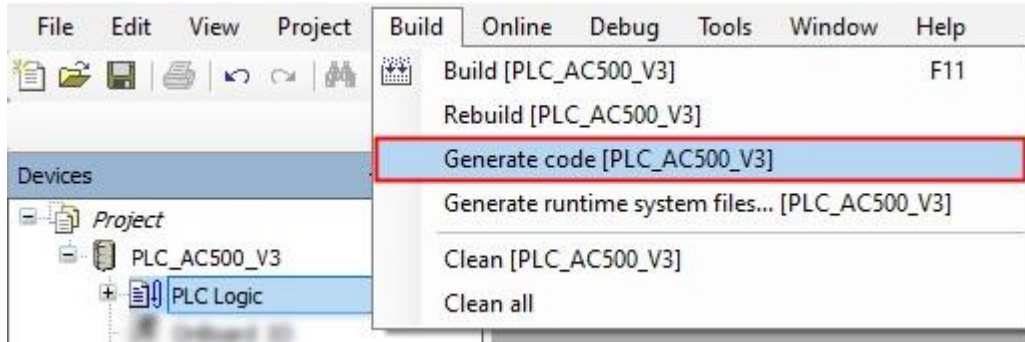
Search object name...

Name	Short Description	Version	Order Number
Action			
Method			
Property			
Transition			

☐ Close this dialog after each transaction ☐ Display all versions

Reset filter Add object Close

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

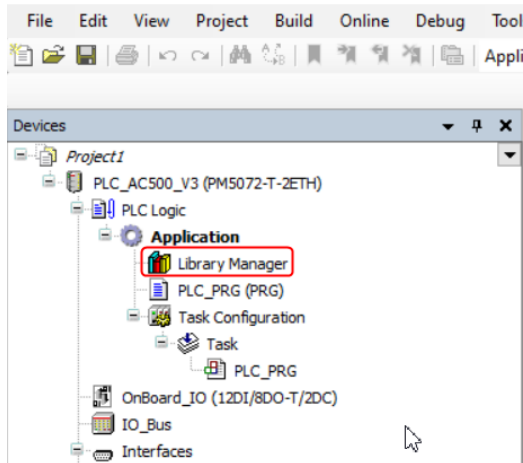


The result of the compiling is shown in the “Messages” field at the bottom of the screen. Select menu “Build” -> “Generate code”.

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

4.8 Library Manager Introduction

The Library Manager is added to Automation Builder Project by default and adding all the relevant default libraries automatically. The library manager offers a wide array of functionality for the user and few of the main functionalities are described here. For all details on the Library Manager and its functionalities, please refer to the latest Automation Builder help file.



To open the Library Manager, user can double click on the same from Automation Builder device tree which opens the library manager in editor’s view.

4.8.1 Add or Search function

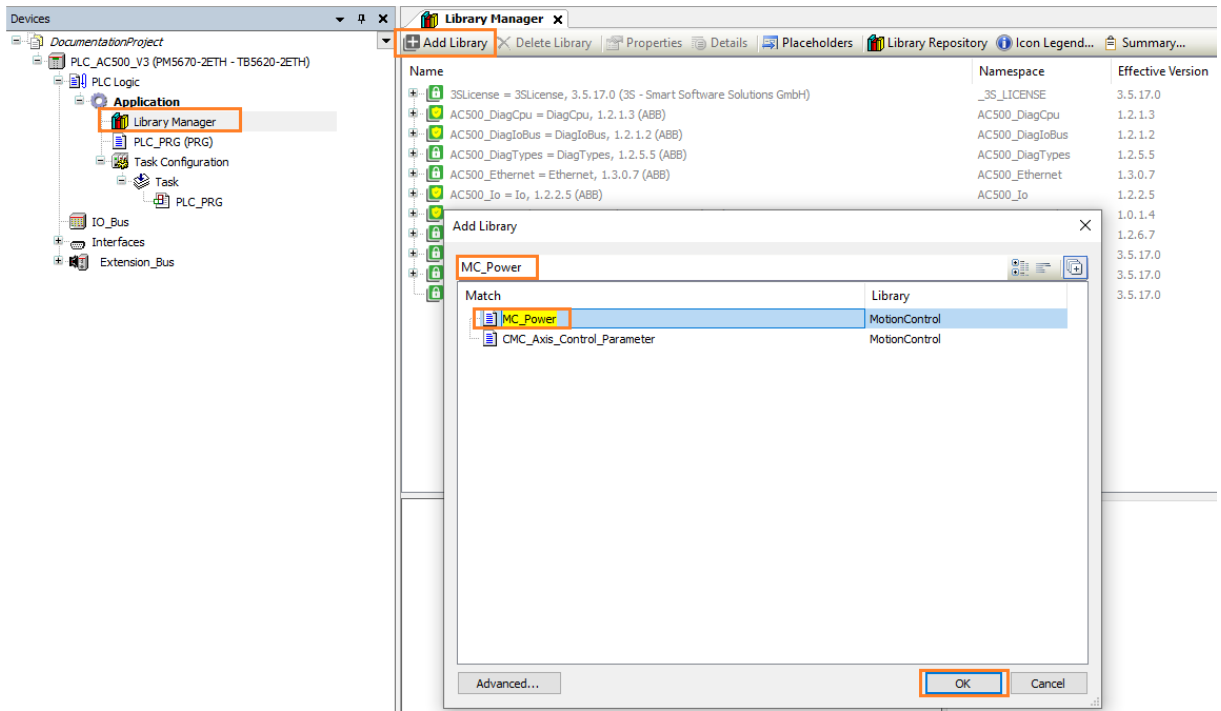
In the Library Manager the search function allows you to quickly find any library or function and add the same to Library manager if this is not yet added.

To search for a library or function:

Click on Add Library

Enter the name of the library or function in the search area, which will list all the possible combinations. Select the function or library and click OK to add the library-to-library manager.

Incase if the library is already added to the library manager, it will show a message saying the same.



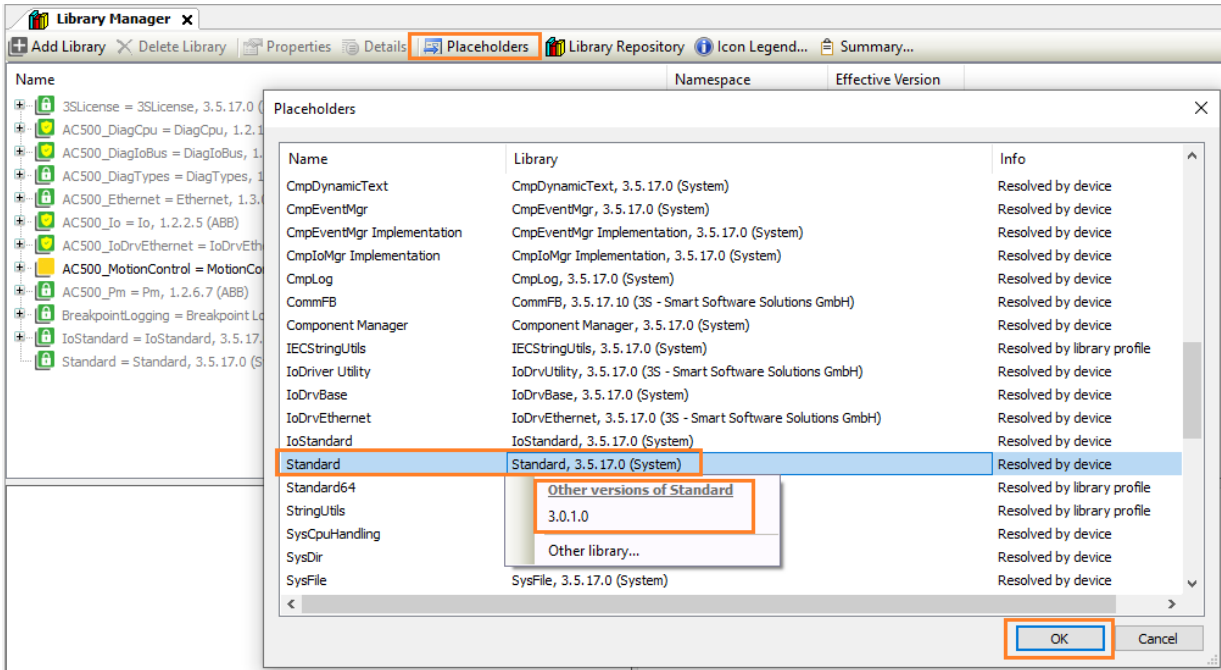
4.8.2 Placeholders and handling different library versions

In the library manager when you add a library it always adds the latest library version referenced in the specified Automation Builder.

When the user has multiple library versions installed, has many Automation Builder versions profiles or if the current project is referencing the wrong library version, the user can change the library version to the correct version using the library placeholder.

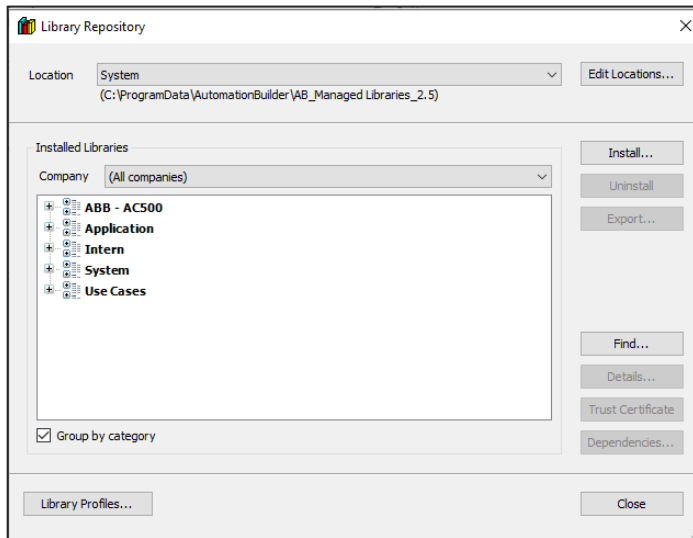
To do this click on the “Placeholders” button which will open the Placeholder window. Double click on a library which needs to be referenced to a different version, this will show all the installed library versions.

Select the library version which user want to use in the project and click OK to update the library version in the current project.



4.8.3 Library Repository

Using Library Repository, user can browse all the installed libraries and its version. Additionally, user can install and uninstall the libraries.



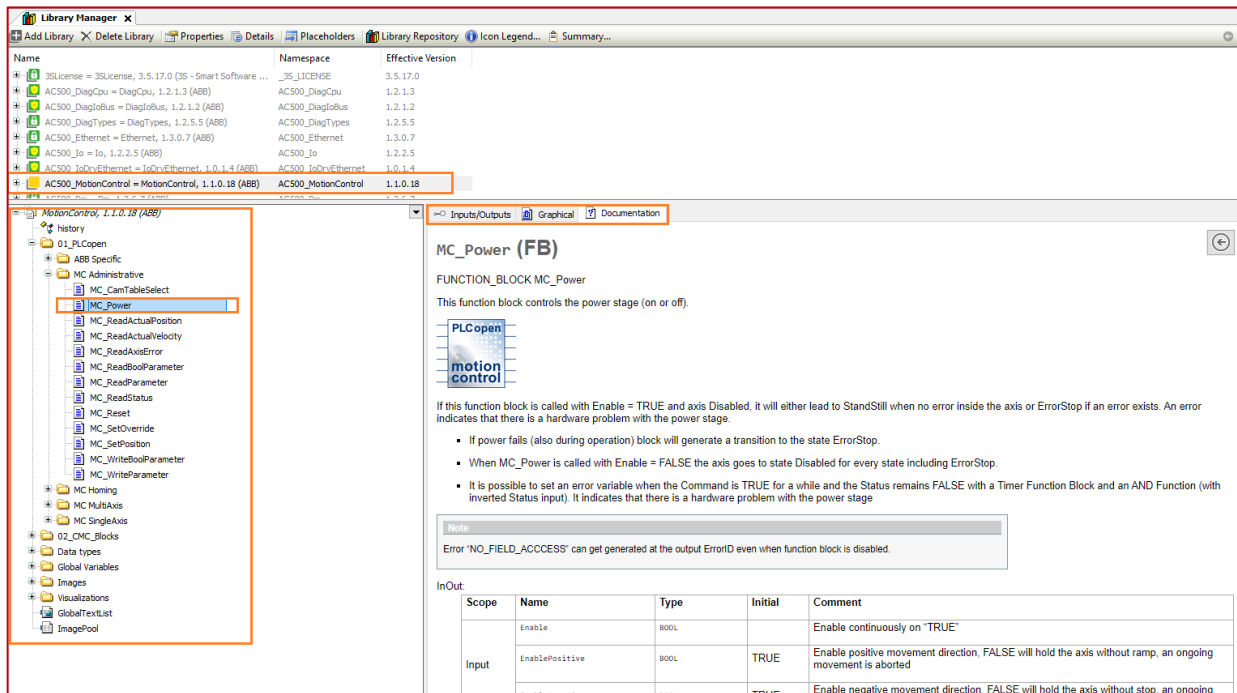
4.8.4 View embedded documentation of all libraries

In the Library Manager you can view embedded documentation of any ABB and 3S libraries.

The full scope of CODESYS library documentation is also available [online](#).

To access any documentation:

- Select a library.
- The contents of the library are shown below.
- From the contents select an object.
- The corresponding documentation is opened on the right side.

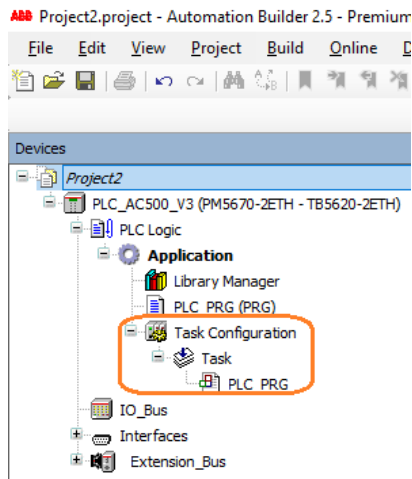


4.9 Task configuration

Tasks are used to define “program calls” for section(s) of code that can be executed in parallel with other tasks. This allows distinct processes to be isolated, hence assisting maintainability.

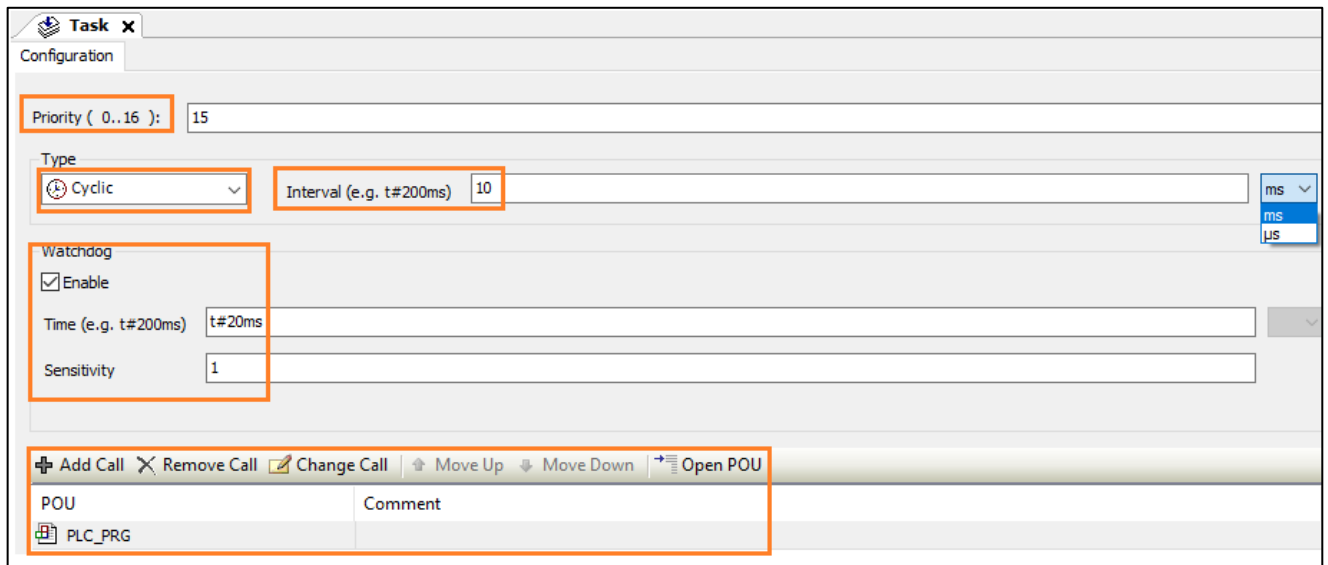
4.9.1 Understanding Task Configuration.

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



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

Double-click “Task” in the device tree -> A tab opens in the editor view.

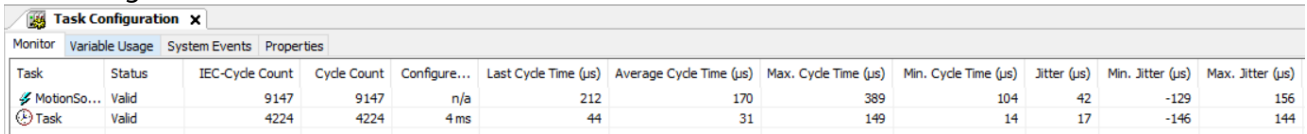


- Priority - This is how the CPU prioritizes the task, when more than one task is defined.
 - Priority 0...15 - Realtime tasks*
 - Priority 16 - Non-Realtime task.
- Type - In the CPU you can run tasks dependent on the demands of the process
- Interval - For cyclic tasks you can set the cyclical execution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

4.9.2 Task types and task monitor

There are different task types configurable based on the application requirement and user can set the task type, time and priority for each task and the program will be called accordingly.

User can monitor all tasks statuses when online with the PLC by double clicking on the Task configuration object on the device tree to open the task monitor window which is showing all configured task and its running status details.



Task	Status	IEC-Cycle Count	Cycle Count	Configure...	Last Cycle Time (µs)	Average Cycle Time (µs)	Max. Cycle Time (µs)	Min. Cycle Time (µs)	Jitter (µs)	Min. Jitter (µs)	Max. Jitter (µs)
MotionSo...	Valid	9147	9147	n/a	212	170	389	104	42	-129	156
Task	Valid	4224	4224	4 ms	44	31	149	14	17	-146	144

This helps the user to monitor task time usage and -statistics in order to set the task parameters to suite the application requirement. It is also possible to read these values from the project during runtime into the project to allow in depth diagnostic e.g. during test and commissioning . After start-up it should be reset once to avoid e.g. max. values from the start-up phase with a right click.

4.10 Real time clock and battery

The real-time clock operates as a computer clock. It saves date and time to a DWORD in DT format

(DATE AND TIME FORMAT), i.e., in seconds passed since the start time: 1 January 1970 at 00:00. If a battery is connected and full, the real-time clock continues to run even if the control voltage is switched off. If no battery is inserted or the battery is empty, the real-time clocks start with the value 0 (=1970-01-01, 00:00:00).

When switching on the control voltage, the system clock of the operating system is set to the value of the real-time clock. The clock can be flexibly synchronized via NTP/SNTP to a network master clock.

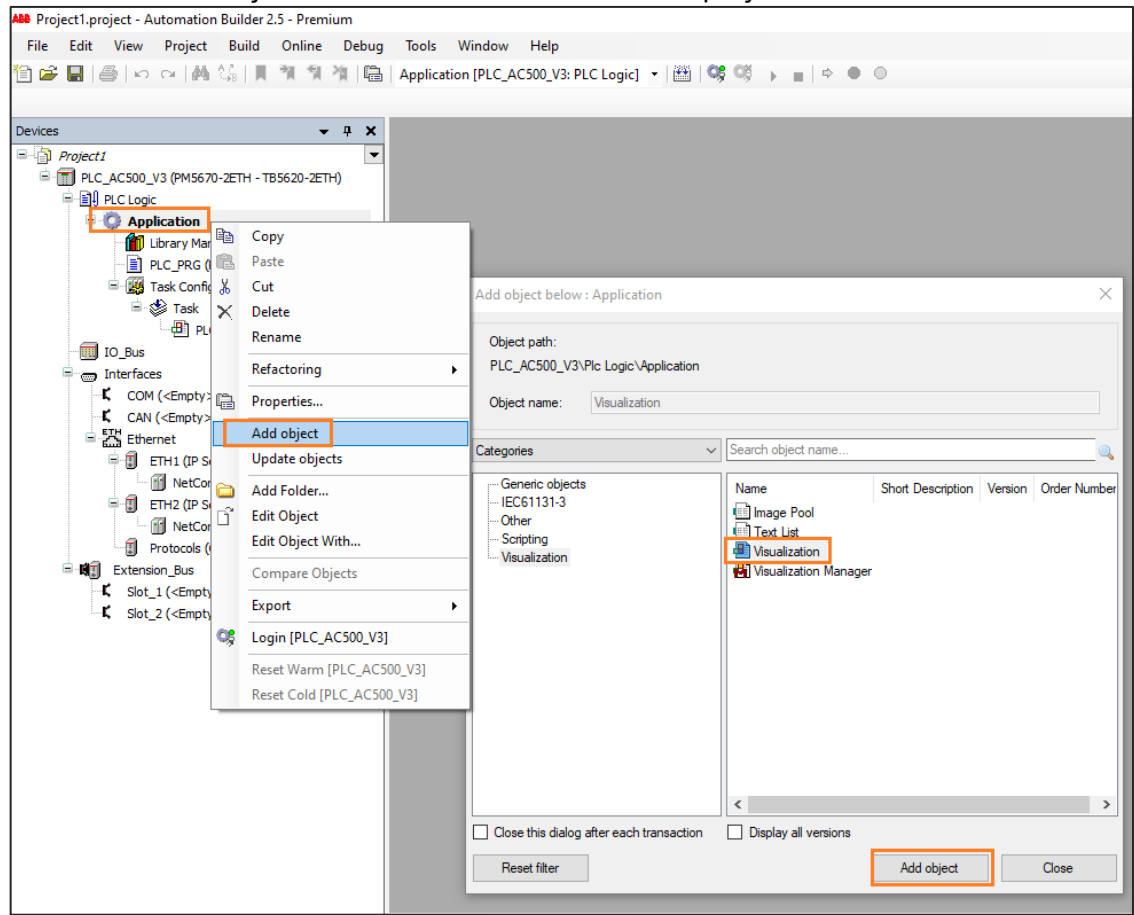
4.11 Integrated project visualization

A visualisation is an integrated visual aide that can be used to create a suitable user interface for your application. The user can link the visualization to the application variables and in this way they can animate and display data. When creating a visualization and an application, you use common functions, for example, as library and source code management or find/replace throughout the project.

4.11.1 Add the Visualization

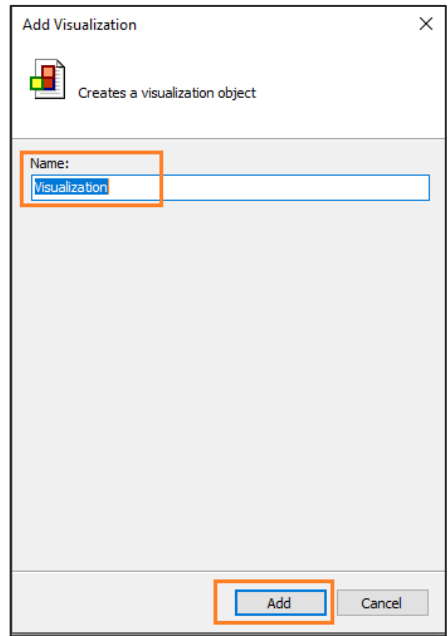
1. Right-click "Application" in the device tree.
2. Select "Add object".
3. Select "Visualization".

4. Select “Add object” to add the Visualization to the project.



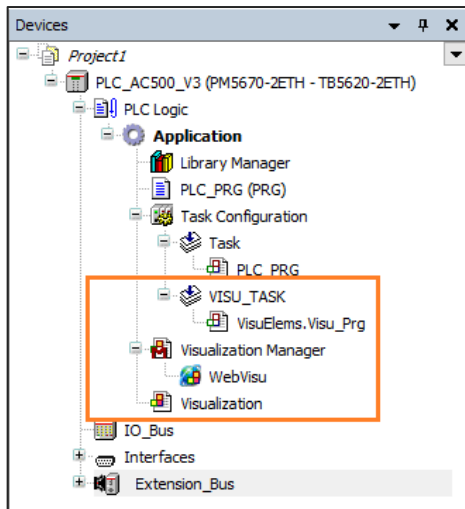
5.

Dialog “Add Visualization” opens.



6. Give a name to visualization and click on “Add”.

You added the objects “Visualization”, “Visualization Manager” and “VISU-TASK” to the device tree.

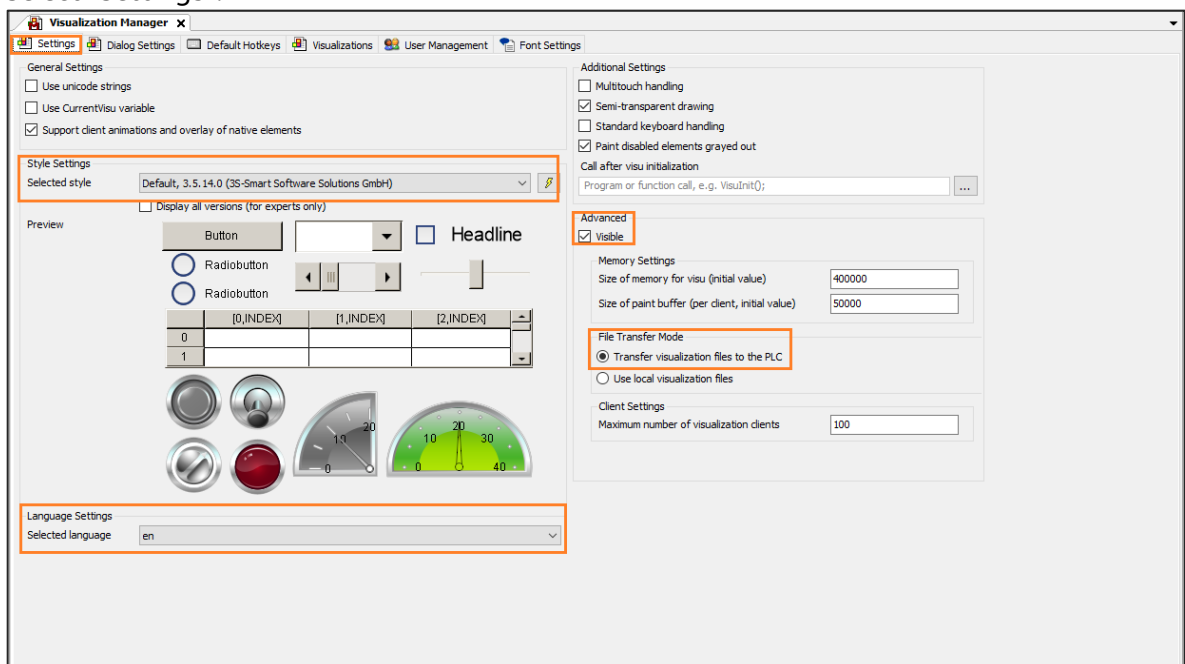


User can add the visualizations in the visualization page and add more visualizations pages as per the requirement.

4.11.2 Set-up the Visualization Manager

The Visualisation manager allows the user to define whether they want to download the visualisation to the PLC's webpage and other settings relating to this.

1. Double-click Visualization Manager in the device tree.
 - A tab opens in the editor view.
2. Select "Settings".



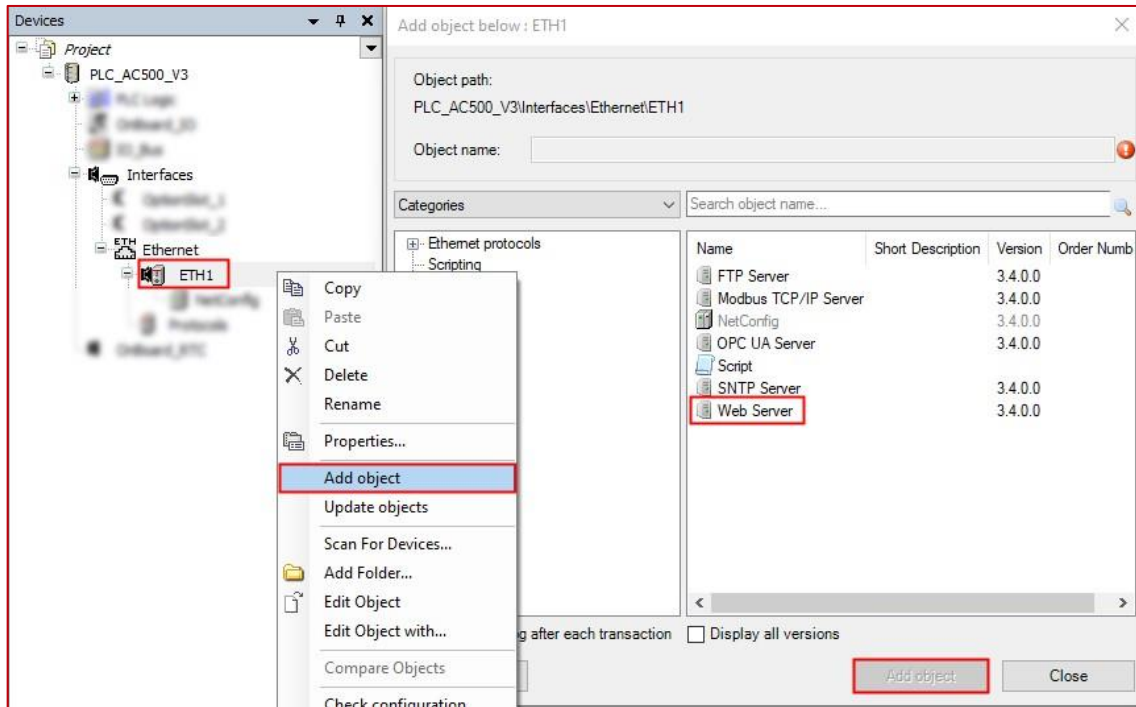
3. Open the drop-down menu "Selected style".
4. Select "Default, x.x.x" (exemplary).
5. Open the drop-down menu "Selected language".
6. Select "en" for English language in the visualization.
7. Enable "Visible" for advanced settings.
8. Keep the file transfer to enable the visualization on the PLC (mandatory for web server function)

4.11.3 Enable web visualization

To add a web server to the project, follow the steps below:

4.11.3.1 Add a web server object to the device tree

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



Right-click “ETH1” in the device tree.

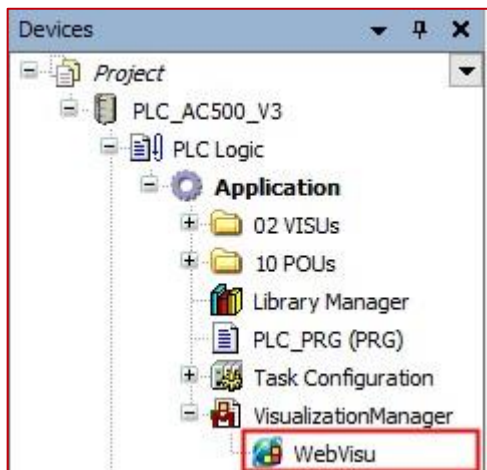
Select “Add object”.

Select “Web Server”.

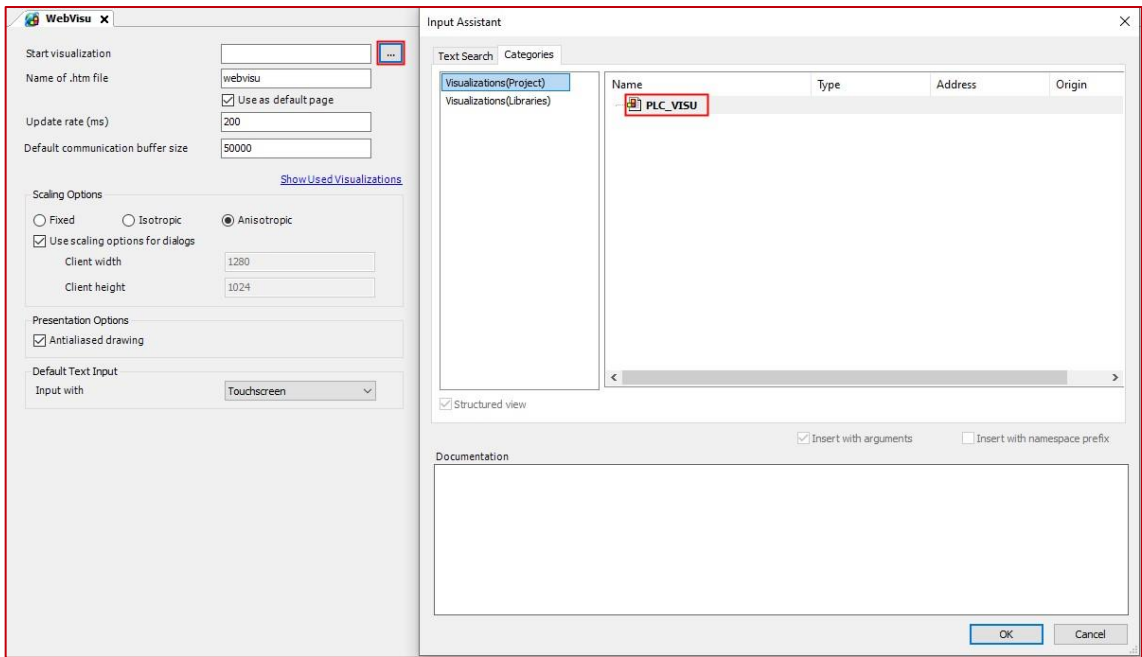
Select “Add object”.

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

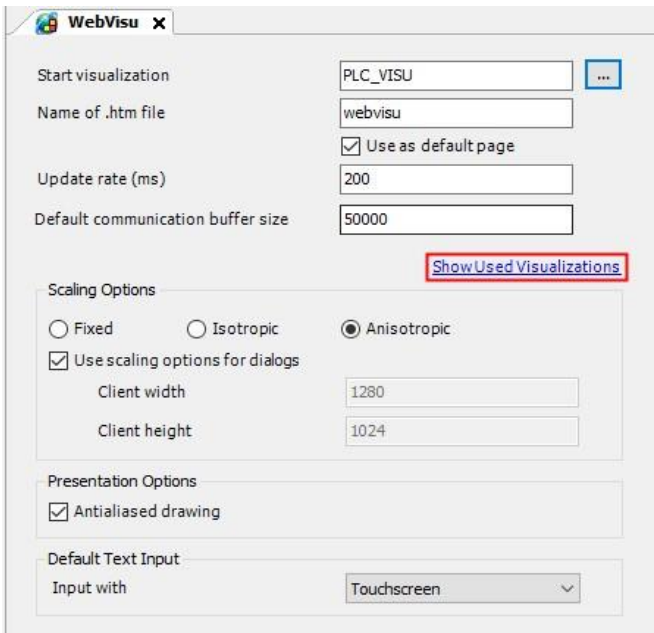
4.11.3.2 Set-up the web server



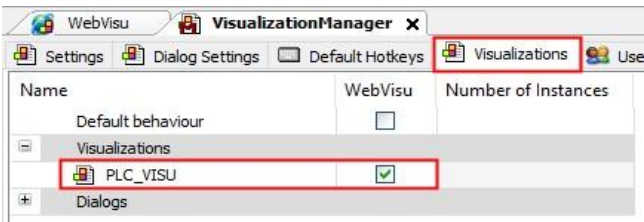
Double-click “WebVisu” in the device tree.



Under “Start Visualization”, select “...”.
A list opens.
Select the “PLC_VISU” screen from the list.
Keep all further settings with default values.



Select the link “Show used visualizations”.



The Visualization Manager editor and there the tab “Visualizations” opens. All screens and dialog elements created in the project are visible.
Here, you can select which screens are enabled or disabled for web visualization.

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



The screenshot shows a Windows File Explorer window with the address bar displaying the path: > This PC > System (C:) > Program Files (x86) > ABB > AutomationBuilder > CODESYS > 2.3.9.55 > Visu. Below the address bar is a table listing files in the directory.

Name	Date	Type	Size	Tags
webvisu.htm	28.04.2010 14:10	Chrome HTML Do...	2 KB	

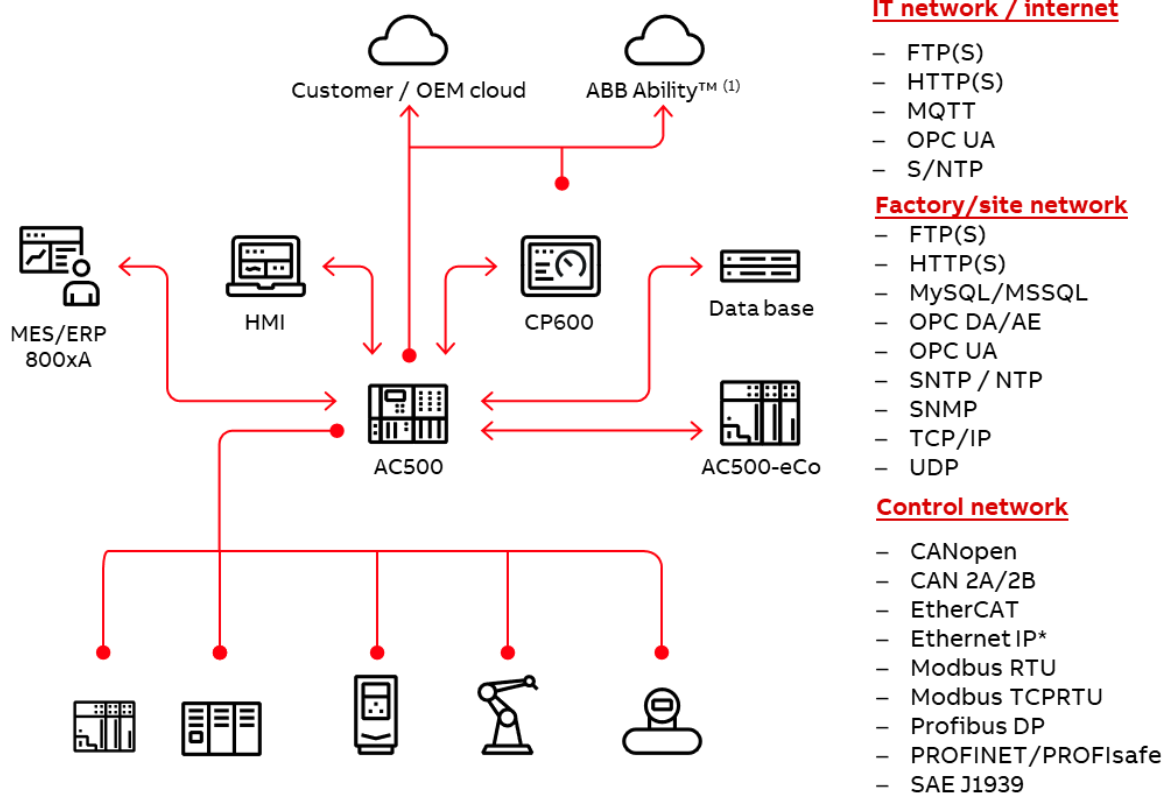
5 AC500 COMMUNICATION PROTOCOLS

5.1 Supported Protocols Overview

The subchapters below focus on the protocols used in motion control applications only: EtherCAT, Modbus RTU and TCP and OPC UA.

AC500 V3 and AC500-eCo V3 have a both lot more of usable protocols for different types of communication needs, either on the onboard ethernet ports or for AC500 also via communication modules (“couplers”) on the left side of the AC500 CPU with additional ports and performance.

Therefore the below picture gives an overview only - for further details and more detailed tabular overview please check the catalog or online help.



5.2 EtherCAT

EtherCAT is currently available in AC500 only via the communication module (coupler) CM579-ETHCAT which is included in the Motion-Kits.

5.2.1 Configuring the CM579-ETHCAT EtherCAT master in the project

This section allows us to configure the behaviour of the Comms module. From here we can define how the hardware will behave. It's important to understand the EtherCAT master once added is split into two parts in the project tree. These two parts are described below.

5.2.2 CM579-ETHCAT

The label will normally be in the format of '[name](CM579-ETHCAT)' this can be changed by the user but is normally left as default. If so the name will be **CM579** (CM579-ETHCAT)

This first part of the EtherCAT master defines the general behavior of the hardware. The following parameters are available:

Parameter	Default value	Value	Description
-----------	---------------	-------	-------------

Run on config fault	No	No	In case of a configuration error, the user program is not started.
		Yes	The user program is started independent of a faulty configuration of the EtherCAT Communication Module.
Max wait run	3000	3000	Maximum wait time for the Master to build up the communication relation to the slaves.
			A restart to build up the communication is initiated as long as BootUpTime has not run out.
Min up-date time	10	0...20000	Priority of the data exchange between CPU and Communication Module.
			This parameter should never be set to values under 10, otherwise important sequences in the CPU might be influenced.
Broken slave behavior	Leave all broken slaves down	Leave all broken slaves down	Broken slaves will not be served.
		Leave addressless slaves down	Only slaves without address will be left down.
		Leave no slaves down	Broken slaves will be ignored.
Distributed clocks	Inaktiv	Inactive	Distributed clocks are inactive.
		Active	Distributed clocks are active.
BootUp-Time	10000	10000	Maximum wait time for the slaves to boot completely. This absolute time value must be a multiple of Max wait run. It defines the time in which Max wait run is restarted to wait for the slaves to boot up completely. The multiplication ratio between Max wait run and BootUpTime can be interpreted as number of trials to boot up the slaves.

In most cases these settings can be left at default but occasionally the user might need to change these to fit the program requirements.

5.2.3 EtherCAT Master Settings

The label will normally be in the format of '[name](ETHCAT-Master)' this can be changed by the user but is normally left as default. If so the name will be **ETHCAT_Master** (ETHCAT-Master)

This second part of the EtherCAT master defines the specific settings that define the behavior of the EtherCAT operation. Here we can set couple of CM579-ETHCAT specific parameters. These parameters will all be set automatically to correct start values, if the motion solution wizard is used

The EtherCAT bus behavior has to be set for real-time/synchronized motion application to "Sync mode 1". With the usage of a synchronize bus behavior, the Task configuration must be configured as "external Event".

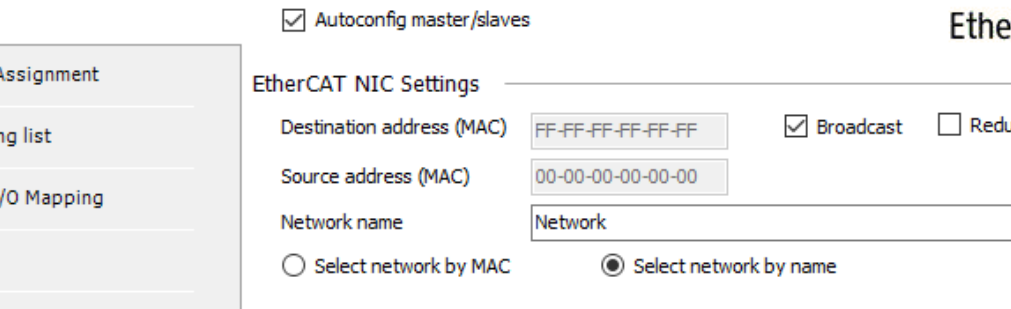
The configuration of EtherCAT modules is based on the device description files for the master and slave devices employed and can be adapted in the project in configuration dialogs. To ensure the simplest and most error-free use possible, we recommend for standard applications that you activate the option for the "Automatic Configuration" of the master, so that the majority of the configuration settings are performed automatically.

The screenshot displays the Siemens STEP 7 HW Config interface. On the left, the project tree shows the configuration for a PLC rack, with 'CM579-ETHCAT' selected. The main area on the right shows the 'CM579-ETHCAT Parameters' table. The table has columns for Parameter, Type, Value, Default Value, Unit, and Description. The 'Bus behavior' parameter is highlighted with a red box, showing a value of 'Sync mode 1'.

Parameter	Type	Value	Default Value	Unit	Description
Run on config fault	Enumeration of BYTE	No	No		Start PLC program even on configuration fault
Broken slave behaviour	Enumeration of DWORD	Leave all broken slaves down	Leave all broken slaves down		Behaviour of broken slaves
Distributed clocks	Enumeration of DWORD	Active	Active		Distributed clocks inactive or active
Bus Target State	Enumeration of BYTE	Operational_OP	Operational_OP		Target state of the EtherCAT bus at application start
Bus behavior	Enumeration of DWORD	Sync mode 1	Asynchronous (IEC bus cycle)		Sync mode 1 - minimum lag (1 bus cycle) between input and output
Optimize I/O update	Enumeration of BYTE	Off	Off		Optimize I/O update

5.2.3.1 “General” Tab

This tab is for the configuration of the basic settings for the EtherCAT master. The preset basic settings originate from the device description file.



ETHCAT_Master x

General

Sync Unit Assignment

I/O mapping list

EtherCAT I/O Mapping

ENI file

☒ Autoconfig master/slaves

EtherCAT

EtherCAT NIC Settings

Destination address (MAC) FF-FF-FF-FF-FF-FF ☒ Broadcast ☐ Redundancy

Source address (MAC) 00-00-00-00-00-00

Network name Network

☐ Select network by MAC ☒ Select network by name

▲ Distributed Clock ▶ Options

Cycle time 4000 μs

Sync offset 20 %

☐ Sync window monitoring

Sync window 1 μs

☒ Master mode (time from first DC slave)

☐ Slave mode (CODESYS time)

☐ External mode (external sync device)

5.2.3.1.1 EtherCAT NIC setting

Autoconfigure Master/Slaves

The auto-configuration mode (“Autoconfig Master/Slaves” option) is activated by default and is adequate for standard applications.

If the mode is not activated, all configuration settings for master and slave(s) must be made manually, for which expert knowledge is required!

The auto-configuration mode option must be switched off for the configuration of slave-to-slave communication.

Even if this option of the master is activated, an expert mode can be activated explicitly for each individual slave that permits the manual editing of the automatically generated process data configuration.

Destination address

MAC address of the device in the EtherCAT network that is to receive telegrams. If set as Broadcast, no “destination address (MAC)” need be specified.

Source address

MAC address of the EtherCAT master device.

Network name

Name or MAC of the network, depending on which of the following options is activated:

Select network by MAC

Network is specified by the MAC-ID. The project then cannot be used on another device, since each network adaptor has a unique MAC-ID.

Select network by Name

Network is identified by the network name and the project is device independent.

5.2.3.1.2 Distributed clock

Cycle time

Time interval after which a new data telegram is dispatched on the bus. If the “Distributed clocks” function is activated in the slave, the master cycle time specified here is transferred to the slave clocks. In this way a precise synchronization of the data exchange can be achieved, which is important if spatially distributed processes require simultaneous actions.

Simultaneous actions are, for example, applications in which several axes must execute coordinated movements at the same time. A very precise, network-wide time base with a jitter of substantially less than 1 microsecond can be achieved in this way.

Sync offset

Parameter for setting the delay time between the DC time base of the EtherCAT Slave and the cycle start of the PLC. With the default value of 20%, the PLC cycle starts 20% of the bus cycle time after the sync interrupt of the slave.

Sync window monitoring

Synchronization of the slaves can be monitored.

Sync window

Time for Sync window monitoring. When the synchronization of all slaves is within this time window, the variable xSyncInWindow (IoDrvEtherCAT) is set to TRUE, otherwise to FALSE.

5.2.3.1.3 Options

Use LRW instead of LWR/LRD

Direct communication from slave to slave is possible. Combined read/write commands (LRW) are used instead of separate read commands (LRD) and write commands (LWR).

Messages per task

Read and write commands (the handling of the input and output messages) can be controlled by means of various tasks.

Automatically restart slaves

The master immediately attempts to restart the slaves in the case of a communication breakdown.

5.2.3.1.4 Master Settings

These settings can be edited only when the “Autoconfig master/slaves” option is deactivated. Otherwise, this is done automatically, and they are not visible here.

Image In Address - First logical address of the first slave for input data.

Image Out Address -First logical address of the first slave for output data.

5.2.3.2 “Sync Unit Assignment” Tab

The tab shows all slaves that are inserted below a specific master with an assignment to the sync units.

With the EtherCAT sync units, multiple slaves are configured into groups and subdivided into smaller units. For each group, the working counter can be monitored for an improved and more exact error detection. As soon as a slave is missing in a sync unit group, the other slaves in the group are also shown as missing. Detection occurs immediately in the next bus cycle because the working counter is checked continuously. With the device diagnostics, the missing group can be remedied as quickly as possible.

Unaffected groups remain operable without any interference.

Device name

Name of the slave

Sync unit

Name of the selected sync unit. You can combine single devices or entire groups (multiple selection) into on sync unit group.

Add

When you type a name in the text field, you can create a new sync unit.

Delete

Deletes the selected sync unit. When slaves are assigned to the group to be deleted, a warning dialog opens. If you click Yes to acknowledge the dialog, then these devices are reassigned to the default group.

5.2.3.3 “IO mapping list” Tab

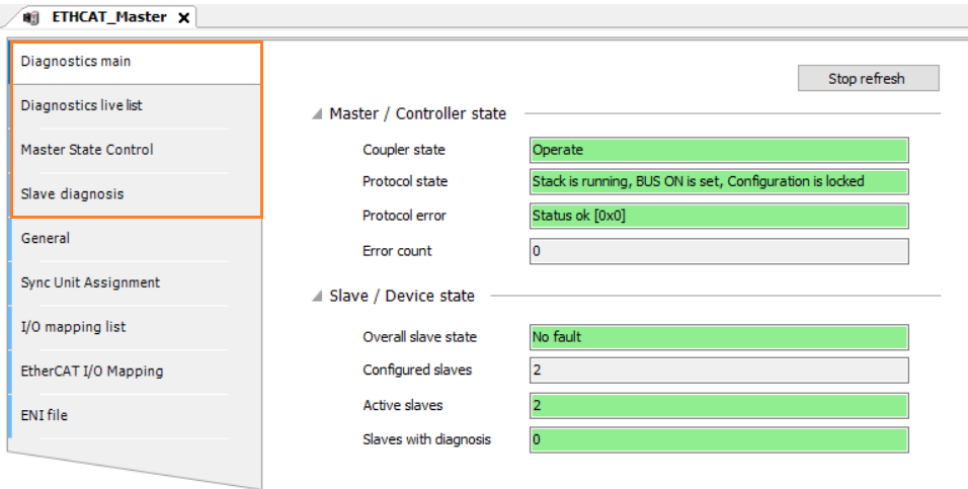
This tab shows all the EtherCAT IO mapping from both EtherCAT master and from all the slaves.

5.2.3.4 ”EtherCAT IO mapping” Tab

This tab shows the working counter variables and this can be used to monitor the EtherCAT bus in the IEC program.

5.2.3.5 Online Diagnosis and State control Tabs

When online with PLC, user gets additional diagnosis tabs which can be used to monitor the EtherCAT master and slave status



5.2.4 EtherCAT Slave Settings

5.2.4.1 General

The basic settings for the EtherCAT Slave are configured on this tab. The basic settings are preset from the device description file.

The screenshot shows the 'General' tab of the 'MicroFlex_e190' configuration window. The left sidebar contains a tree view with the following items: General (selected), Expert Process Data, Process Data, Startup Parameters, EoE Settings, EtherCAT I/O Mapping, I/O mapping list, Status, and Information. The main area is divided into several sections:

- Address:**
 - AutoInc address: 0
 - EtherCAT address: 1001
- Additional:**
 - ☒ Expert settings
 - ☐ Optional
- Distributed Clock:**
 - Select DC: DC-Synchronous
 - ☒ Enable: 4000 Sync unit cycle (µs)
- Sync0:**
 - ☒ Enable Sync 0
 - ☒ Sync unit cycle: x 1, 4000 Cycle time (µs)
 - ☐ User-defined: 0 Shift time (µs)
- Sync1:**
 - ☐ Enable Sync 1
 - ☒ Sync unit cycle: x 1, 4000 Cycle time (µs)
 - ☐ User-defined: 0 Shift time (µs)
- Startup Checking:**
 - ☒ Check vendor ID
 - ☒ Check product ID
 - ☐ Check revision number
 - ☐ Download expected slot configuration
- Timeouts:** (Collapsed)
- DC Cyclic Unit Control: Assign to Local µC:**
 - ☐ Cyclic unit
 - ☐ Latch unit 0
 - ☐ Latch unit 1
- Watchdog:**
 - ☐ Set multiplier (Reg. 16#400): 2498
 - ☒ Set PDI watchdog (Reg. 16#410): 1000 = 100.00 ms
 - ☐ Set SM watchdog (Reg. 16#420): 1000 = 100.00 ms
- Identification:**
 - ☒ Disabled
 - ☐ Configured station alias (ADO 0x0012): Value 1001
 - ☐ Explicit device identification (ADO 0x0134)
 - ☐ Data Word (2 Bytes): ADO (hex) 16#0

Address

Fields can be edited only when the auto-configuration mode of the EtherCAT Master is disabled.

“AutoInc address”

Self-incrementing address (16-bit) that results from the position of the slave in the network. The address is used only during the system boot process when the master assigns the EtherCAT addresses to its slaves. When the first message runs through all the slaves for this purpose, each slave increments its AutoInc address by 1. The slave with address 0 then gets the data. A possible input here is “-8”.

EtherCAT address

Final address of the slaves, assigned by the master during bootup. The address is independent of the position of the slave in the network.

5.2.4.1.1 Additional

Expert settings

Additional settings are possible for the startup checking and time monitoring (see below). The Expert Process Data tab is also available in the device editor to add additional PDO mapping.

However, expert settings are not required for standard applications. The auto-configuration mode is recommended and sufficient for standard applications.

Optional

At the start of the stack, the system checks whether optional devices are available.

The slave is defined as optional and no error message is generated if the device is missing from the bus system. If a device is not found, then it is disabled automatically and displayed in gray in the device tree. A corresponding message is displayed in the logger.

Note: If you define a slave as “optional”, then it has to have a unique identification. You can change this by means of the three possible settings in the Identification section.

Available only when the “Autoconfig master/slaves” option is selected in the settings of the EtherCAT Master and the EtherCAT Slave supports this function.

5.2.4.1.2 Distributed Clock

Select DC

List box with all settings for the distributed clocks of the device description file

Enable

Cycle time for the data exchange. It is displayed in the Sync unit cycle (μs) input field and determined by the cycle time of the master. As a result, the master clock can synchronize the data exchange in the network.

5.2.4.1.3 Sync0 and Sync1

The Sync0 and Sync1 settings described here are slave dependent.

Enable Sync 0/1

Synchronization unit Sync0/1 is used. A synchronization unit describes a set of process data that is exchanged synchronously.

Sync unit cycle

The master cycle time (multiplied by the factor selected from the list box) is used as the synchronization cycle time for the slave. Cycle time (μs) displays the cycle time currently set.

User-defined

A custom cycle time (in microseconds) can be specified in the Cycle time (μs) field.

5.2.4.1.4 Diagnosis

This area section appears in online mode only.

Current State

State of the slave. Possible states: Init, Preoperational, Safe Operational, and Operational

The state Operational indicates that the slave configuration has been correctly completed and that process data (inputs and outputs) are being accepted.

5.2.4.1.5 Start-up Checking

Check vendor ID and Check product ID

By default, the vendor ID and product ID of the device are checked against the current configuration settings when the system boots up. If they do not agree, then the bus is stopped, and no further actions are executed. This is done to prevent an incorrect configuration from being loaded onto the bus system.

Options for deactivating the corresponding check.

Check revision number

The revision number is checked during the system startup according to your selection in the list box.

Download expected slot configuration

For online verification of the configured and actual module configuration. If the configurations do not match, then the device still switches to "Run". In this case, an entry is made in the device logbook.

5.2.4.1.6 Timeouts

By default, watchdog is not defined for the following actions. If necessary, an appropriate timeout can be specified here (in milliseconds):

SDO access

Transmits the SDO list at system start. Specified in milliseconds.

I -> P

Switch from Init mode to Preoperational mode. Specified in milliseconds.

P -> S / S -> O

Switch from Preoperational mode to Safe Operational mode, or from Safe Operational mode to Operational mode. Specified in milliseconds.

5.2.4.1.7 DC Cyclic Unit Control: Assign to Local µC

One or more options for the Distributed Clock function can be activated here that should be used on the local microprocessor. The check is performed in the registry at 0x980 in the EtherCAT Slave. Possible settings:

Cycle unit

Latch unit 0

Latch unit 1

5.2.4.1.8 Watchdog

Set multiplier

The PDI watchdog and SM watchdog receive their impulses from the local terminal clock divided by the watchdog multiplier.

Set PDI watchdog

This watchdog triggers when there is no PDI communication with the EtherCAT Slave controller for longer than the PDI (Process Data Interface) watchdog time which has been set and activated.

Set SM watchdog

This watchdog triggers when there is no EtherCAT process data communication with the terminal for longer than the SM (Sync Manager) watchdog time that has been set and activated.

5.2.4.1.9 Identification

In this section, you set the device identification of the slave. As a result, you can make the address of the slave independent of its position in the bus.

The following options are visible only when the Activate expert settings option or Optional option is selected.

If you have identified the slave as Optional, then you have to assign a unique ID to it.

Disabled

The identification of the slave is not checked.

Configured station alias (ADO 0x0012)

Address that is stored in the EEPROM of the device.

You can change the value in the Scan Devices dialog or in online mode. For stock devices, you need to assign this number one time. This means that you have to connect the device one time to an EtherCAT Master and save the number.

Write to EEPROM

Visible in online mode only for Configured station alias. Writes the defined address for Value to the EEPROM of the slave.

Explicit device identification (ADO 0x0134)

The device identification is hard set on the hardware (for example, by DIP switches). It is displayed in Actual address.

Data Word (2 Bytes)

A 2-byte value for the identification is saved in the slave.

Value

Expected value for the check. If the actual value does not correspond to this setting, then an error is issued.

ADO (hex)

Initial value from the device description. You can change this value in the Data word option.

Actual address

Visible in online mode only. Displays the address of the slave. You can use this display for checking the success of the Write to EEPROM command.

5.2.4.2 FMMU/Sync

The tab shows the FMMUs and Sync Manager of the EtherCAT Slave as they are defined in the device description file. There is an option to edit the FMMUs and Sync Manager (for example, for the configuration of slave-to-slave communication).

Requirement: The auto-configuration mode in the EtherCAT Master is disabled.

Note that these are expert settings which are not usually required for standard applications.

5.2.4.3 Expert Mode Process Data

The tab provides another more detailed view of the process data, which is also displayed in the Process Data dialog. Moreover, the download of the PDO assignment and the PDO configuration is enabled here.

Requirement: The expert settings for the slave are selected.

5.2.4.3.1 Sync Manager

List of the Sync Managers with data size and PDO type

5.2.4.3.2 PDO assignment (16#1C12/16#1C13)

List of the PDOs assigned to the selected Sync-Manager.

When a check box is selected, the PDOs are enabled and I/O channels are created. This is similar to the simple PDO configuration view.

5.2.4.3.3 PDO list

List of the PDOs assigned to the selected Sync-Manager.

You can add new entries or edit or delete existing entries by executing the respective commands (Add, Delete, Edit) in the command bar or context menu.

Edit PDO list

Name

Index

Direction

TxPDO (input): The PDO is transmitted from the slave to the master.

RxPDO (output): The PDO is transmitted from the master to the slave.

Flags

Mandatory: The PDO is required and cannot be disabled in the PDO assignment.

Fixed content: The contents of the PDO are fixed and cannot be modified. It is then not possible to add entries in PDO contents.

Virtual PDO: Reserved for future use

Exclude PDOs

It is possible to define an exclusion list. When a PDO is enabled in the PDO assignment, others are disabled and cannot be enabled.

Sync unit

ID of the Sync Manager to which the PDO is to be assigned

5.2.4.3.4 PDO Contents

Displays the contents of the PDOs selected in the PDO list. You can add new entries or edit or delete existing entries by executing the respective commands (Add, Delete, Edit) in the command bar or context menu. You can change the PDO order by clicking Move Up and Move Down.

Note: That you only need to use this if the PDO option you require is not listed in the Axis objects Mapping Tab.

5.2.4.3.5 Download

PDO assignment

Specific CoE commands for initializing the 0x1cxx objects are generated and written to the slave.

PDO configuration

The CoE commands for 0x16xx or 0x1axx are generated, and then the PDO mapping is downloaded to the slave. Normally, the default values originate from the ESI file and the device has to support this functionality. For example, if a device has a fixed configuration, then these commands are regarded as flawed.

Load PDO info from the device

The current PDO configuration is read from the slave and entered into the configuration. The lists in the upper and lower right are then deleted and filled with the read data. This is especially useful when the ESI file is incomplete and the configuration is available only on the slave.

5.2.4.4 Process Data

The tab of the EtherCAT configurator displays the process data for the inputs and outputs of the slave. The data is preset from the device description file.

Select the Outputs

The table shows the outputs of the slave defined by Start address, Type, and Index.

If outputs of the device are enabled here (for writing), then these outputs can be assigned to project variables in the EtherCAT I/O Mapping dialog.

Select the Inputs

The table shows the inputs of the slave defined by Name, Type, and Index.

If inputs of the device are enabled here (for reading), then these inputs can be assigned to project variables in the EtherCAT I/O Mapping dialog.

5.2.4.5 Startup Parameters

On the tab, the SDOs (service data objects) for 'CAN over EtherCAT' (CoE) or the IDNs (identification numbers) for 'Servodrive over EtherCAT' (SoE) are defined for the current slave. These parameters are determined for the device when the system is started.

The object directory with the required data objects is described in the EtherCAT XML description file or in an EDS (Electronic Device Description) file that is referenced in the XML file.

Requirement: The device supports 'CAN over EtherCAT' or 'Servodrive over EtherCAT'.

Some modules that are inserted below a slave have their own startup parameters. These parameters are also displayed in this list but cannot be edited here. The parameters are modified in the editor of the corresponding module.

5.2.4.6 EoE Settings

This tab is used to configure the communication settings for the individual slaves that support Ethernet over EtherCAT (EoE).

Note: EoE is supported by ABB Servo Drives MicroFlex e190 and MotiFlex e180.

5.2.4.6.1 Settings

Virtual Ethernet Port

Enables the EOE (Ethernet Over EtherCAT) functionality of the slave. A unique Virtual MAC ID has to be defined.

Virtual MAC ID

Input field for the Virtual MAC ID

Switch port

The device acts as a switch. No additional network settings are required.

IP port

The device acts as an IP port. The IP Settings have to be configured.

5.2.4.6.2 IP Settings

The Ethernet communication parameters have to be set according to the parameters of the virtual Ethernet adapter.

IP address

IP address of the slave in the network (length: 4 bytes)

The IP port has to be in the same range as the virtual Ethernet adapter. For example, if the address of the network adapter is 192.168.1.1 and the subnet mask is 255.255.255.0, then the IP port has to be in the range from 192.168.1.2 to 192.168.1.254.

Subnet mask

Subnet mask (length: 4 bytes)

Default gateway

Default gateway (length: 4 bytes)

DNS (Domain Name Services) server

IP address of the DNS server

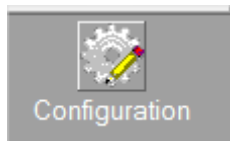
DNS name

Name of the DNS server

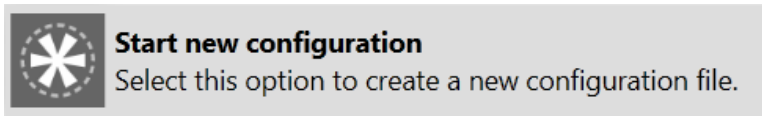
5.2.5 Setting up the PLC and ABB Servo EtherCAT Slave for EoE Comms

5.2.5.1 How to configure e1x0 drive

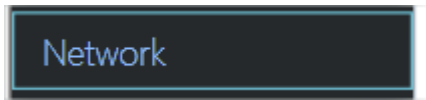
1. Open workbench
2. Connect to drives over USB!
3. Go Online
4. Configuration



5. Start new configuration



6. Network



- Set up the standard Ethernet E1 to be on a different network subnet and IP address to that of Ethernet network we are connecting to. Eg 192.168.10.1

Host port (E1)

DHCP Enabled ☐

Address

Mask

Gateway

EtherCAT EoE port (E2)

Address

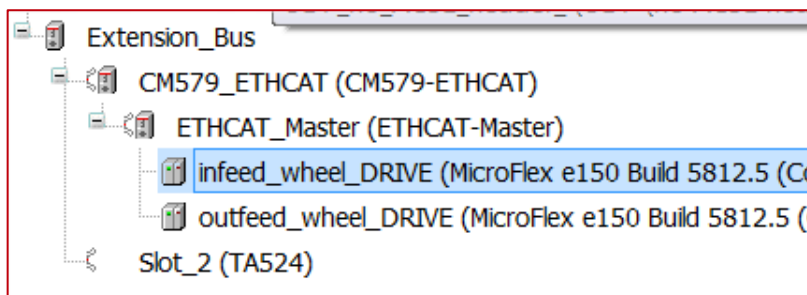
Mask

Gateway

- Set up E2 EoE to be on the same network subnet and IP to that of Ethernet network we are connecting to. Eg 192.168.0.1
- Apply these changes to the Drive using the Apply button
- Do the same for all other drives but *indexing* the IP address in each case.

5.2.5.2 How to configure Automation Builder Project

- Open Automation Builder and set up your EtherCAT network as per other APP notes
- Select the EtherCAT drive you have configured for the EtherCAT slave list



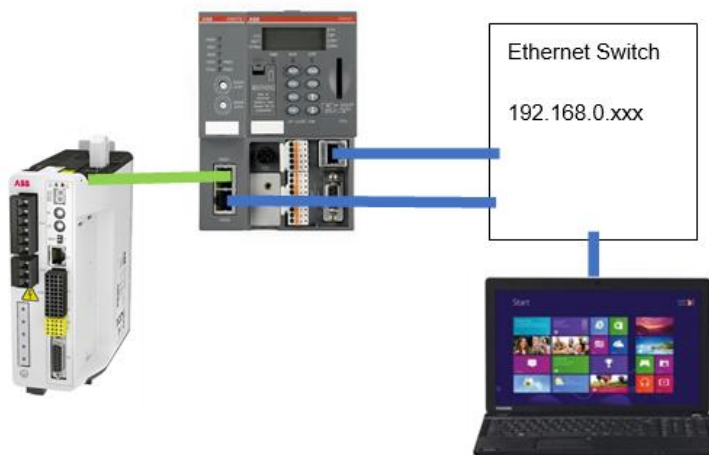
- Select EoE Settings
- Tick the box for Virtual Ethernet Port
- Enter the same settings you have defined above in the drive.

Slave	Expert Process Data	Process Data	Startup parameters	EoE settings
Settings				
<input checked="" type="checkbox"/> Virtual Ethernet Port				
Virtual MAC Id:		02-01-05-10-03-E9		
<input type="radio"/> Switch Port		<input checked="" type="radio"/> IP Port		
IP Settings				
IP Address:		192 . 168 . 0 . 1		
Subnet Mask		255 . 255 . 255 . 0		
Default Gateway:		192 . 168 . 0 . 254		
DNS Server:		0 . 0 . 0 . 0		
DNS Name:		infeed_wheel_DRIVE		

6. Complete and Build your project
7. Open up CODESYS and Download project

5.2.5.3 How to connect your Network

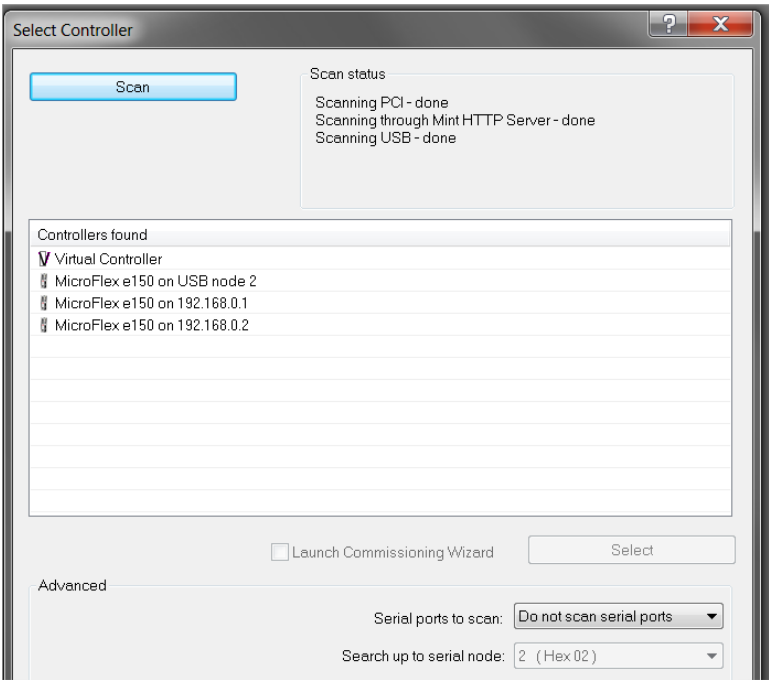
For our example we will configure a remote I/O system that looks like this:



5.2.5.4 How to use this configuration

Once we have a running configured network (set up as shown above) we can use it to connect to the drives on the network,

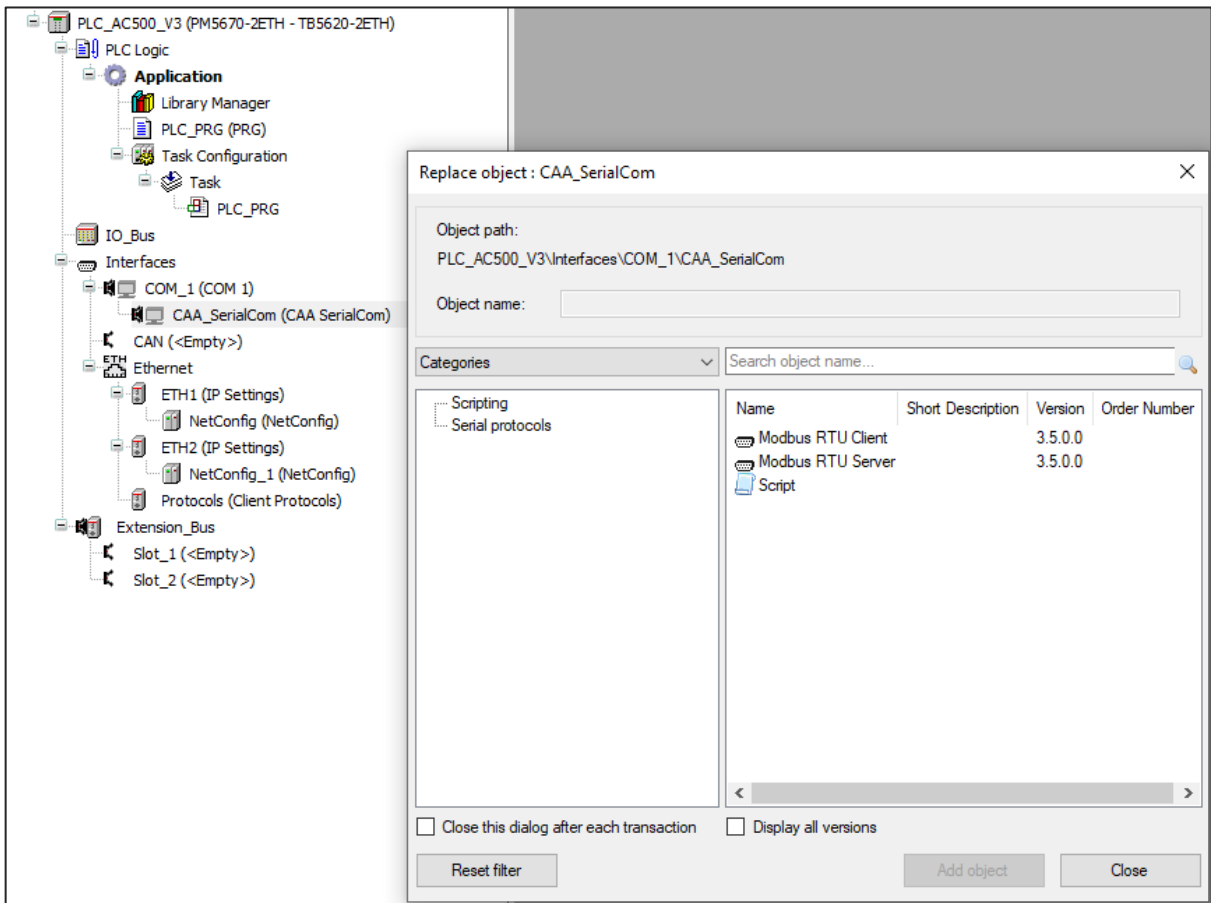
1. but first we must turn **OFF our PC's firewall** so workbench can scan the connected drives.
2. Once this is done, we can reopen workbench and we should see that our configured drives have appeared.
3. We can connect to the drives now and configure them as normal.



Modbus RTU

The Modbus RTU protocol is implemented in the AC500 Processor Modules. Modbus RTU is a master-slave protocol. The Master sends a request to the Slave(s)and receives the response(s). The Modbus operating mode of a serial interface is set in the PLC configuration.

To enable Modbus RTU on a serial, interface the protocol setup per default must be replaced by either Modbus RTU Client (Master) or Server (Slave), depending on required operation mode.



A serial interface supports only one protocol/operation mode at once.

5.2.6 How to add a Serial Protocol

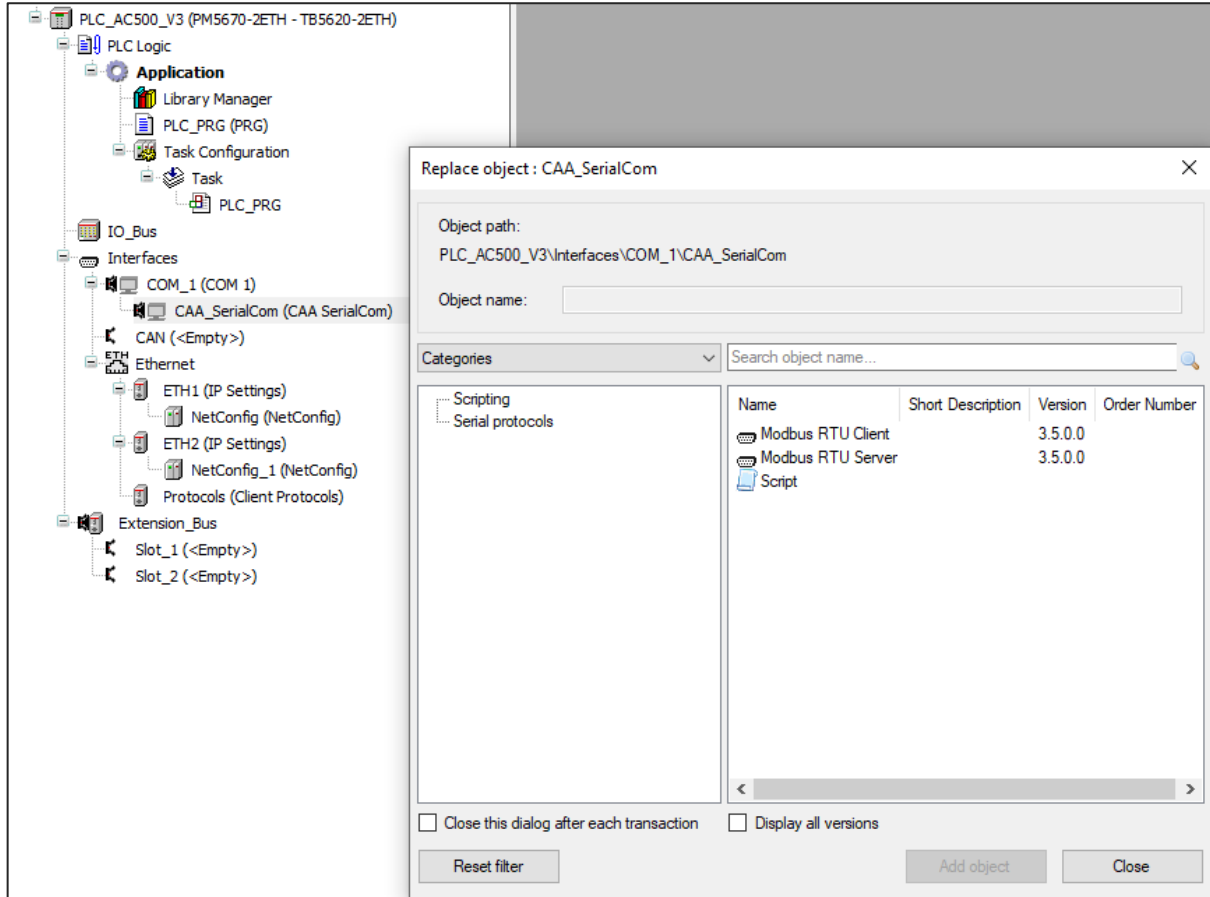
Right click node “COM (<Empty>)” and click “Add object”

Select COM 1 and click on Replace object. This will add COM_1 and “CAA_SerialCom” node.

Right-click node “CAA_SerialCom” and click “Add object”.

Select “Modbus RTU Client” or “Modbus RTU Server” and click “Add object”.

“CAA_SerialCom” is replaced by your selection.



Serial parameters to be set selecting the interfaces node COM_1. They are common for both operating modes client and server.

Parameter	Type	Value	Default Value	Unit	Description
Run on config fault	Enumeration of BYTE	No	No		Start PLC program even on configuration fault
Baudrate	Enumeration of DWORD	19200	19200	Bits/s	Set the baudrate in Bits per seconds
Parity	Enumeration of BYTE	None	None		Set the parity Bit type
Data Bits	Enumeration of BYTE	8	8	Bits/character	Set the character size
Stop Bits	Enumeration of BYTE	1	1		Set the number of stop Bits per character 2 means 1,5 when character size is 5 Bits
Flow control	Enumeration of BYTE	No flow control	No flow control		Flow control
PLC Boot parameters					
Serial communication node	Enumeration of BYTE	RS-232	RS-232		The serial port can be configured to work according to the RS-232 or the RS-485 serial transmission standard

The parameter “Data bits” always has to be set to “8” for Modbus.

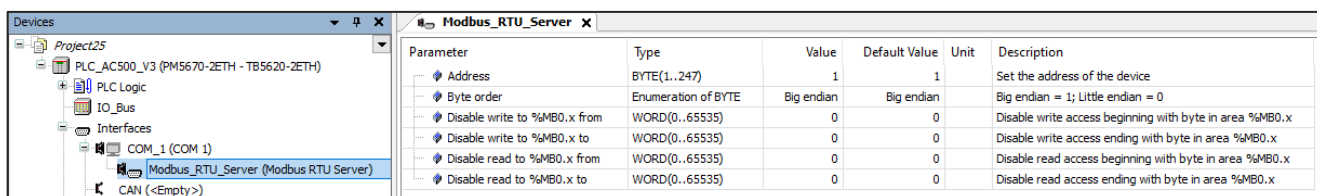
To realize the Modbus client functionality, user need to use the function blocks in the application program.

Note: For the detailed information on Modbus RTU communication please refer the Automation Builder online help file.

5.2.7 Modbus RTU Server (Slave)

In this operating mode, *no Function Block is required for Modbus communication*. Sending and receiving Modbus telegrams is performed automatically as controlled by the communications Master.

Server specific parameters to be set selecting the protocol's node "Modbus_RTU_Server".



Parameter	Type	Value	Default Value	Unit	Description
Address	BYTE(1..247)	1	1		Set the address of the device
Byte order	Enumeration of BYTE	Big endian	Big endian		Big endian = 1; Little endian = 0
Disable write to %MB0.x from	WORD(0..65535)	0	0		Disable write access beginning with byte in area %MB0.x
Disable write to %MB0.x to	WORD(0..65535)	0	0		Disable write access ending with byte in area %MB0.x
Disable read from %MB0.x from	WORD(0..65535)	0	0		Disable read access beginning with byte in area %MB0.x
Disable read to %MB0.x to	WORD(0..65535)	0	0		Disable read access ending with byte in area %MB0.x

Address: Bus address of the PLC as Modbus RTU Server on that interface

Byte Order:

Format/Endianness for the transmission of WORD values (register) within the request/response telegram (default: Big Endian)

Disable:

Parameter	Default	Value	Description
Disable write to %MB from	0	0 ... 65535	Disable write access starting at %MBx
Disable write to %MB to	0	0 ... 65535	Disable write access up to %MBx
Disable read from %MB from	0	0 ... 65535	Disable read access starting at %MBx
Disable read from %MBx to	0	0 ... 65535	Disable read access up at %MBx

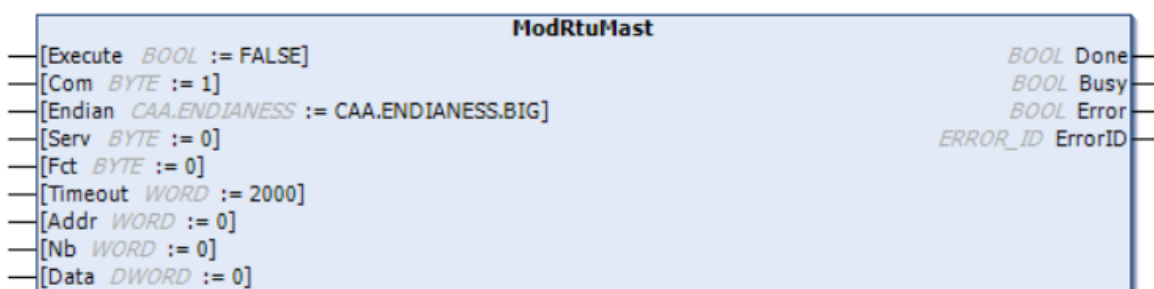
It is possible to disable read and/or write access to individual segments. Reading/writing is disabled beginning at the set start address and is valid up to the set end address (inclusive).

In this operating mode, no Function Block is required for Modbus communication. Sending and receiving Modbus telegrams is automatically.

5.2.8 Modbus RTU Client (Master)

"Modbus RTU Client" does not have any protocol parameters. In this operating mode, the telegram traffic with the server(s) is handled via the Function Block. This Function Block sends Modbus request telegrams to the server(s) via the set interface and receives Modbus response telegrams from the server(s) via this interface.

The Modbus Client functionality then must be realized with Function blocks (such as ModRtuMast) in the application program.

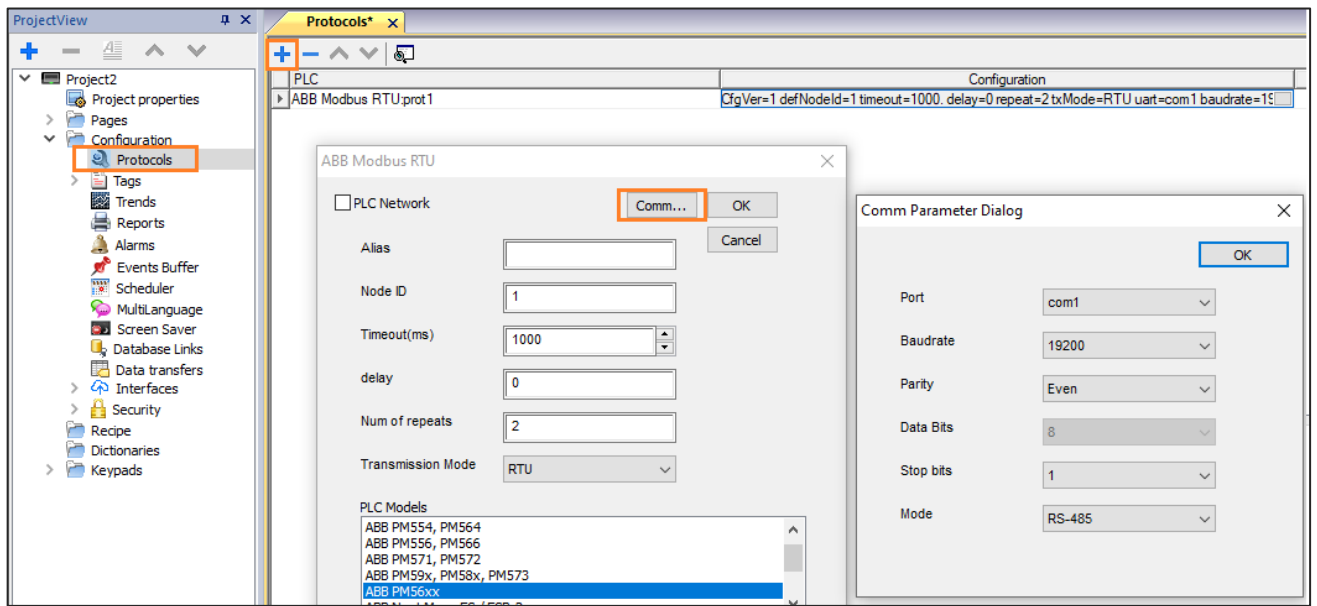


5.2.9 HMI Modbus RTU communication

5.2.9.1 CP600 HMI Modbus RTU communication

For CP600 HMI configuration, user must have installed Panel Builder software using Automation Builder Installation manager.

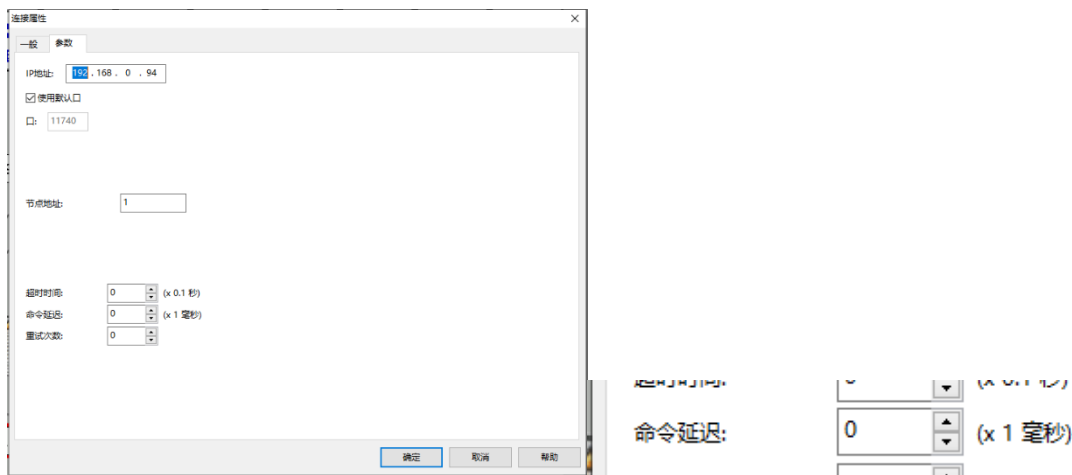
For CP600 communication with an AC500 PLC the internal “AC500 / Codesys protocol” is recommended. If the CP600 panel HMI should communicate via Modbus, user need to configure AC500 COM port as ModbusRTU Server and the respective protocol selection and configuration needs to be done on the Panel builder software to establish the communication successfully.



More details on the CP600 configuration and communication with the AC500, please refer to the latest panel builder help file.

5.2.9.2 3rd Party HMI's and Modbus RTU communication

If the user is using a 3rd Party HMI, its advised to take great care when using CoDeSys protocols. These 3rd party protocols sometimes have no data flow control and can poll the PLC too fast so that they can overload the communication line (“send next when finished” which means PLC is always sending when no delay is entered). If using this protocol is unavoidable then check the settings for things such as ‘Command delay’ which can be used to add delays between Comms events.



5.3 Modbus TCP/IP

The Modbus TCP protocol is implemented in the AC500 Processor Modules. Modbus is a master/slave (client-server) protocol. The client sends a request to the server(s) and receives the response(s).

Each Ethernet interface (ETH1, ETH2 etc.) can work as Modbus client and server interface in parallel if required.

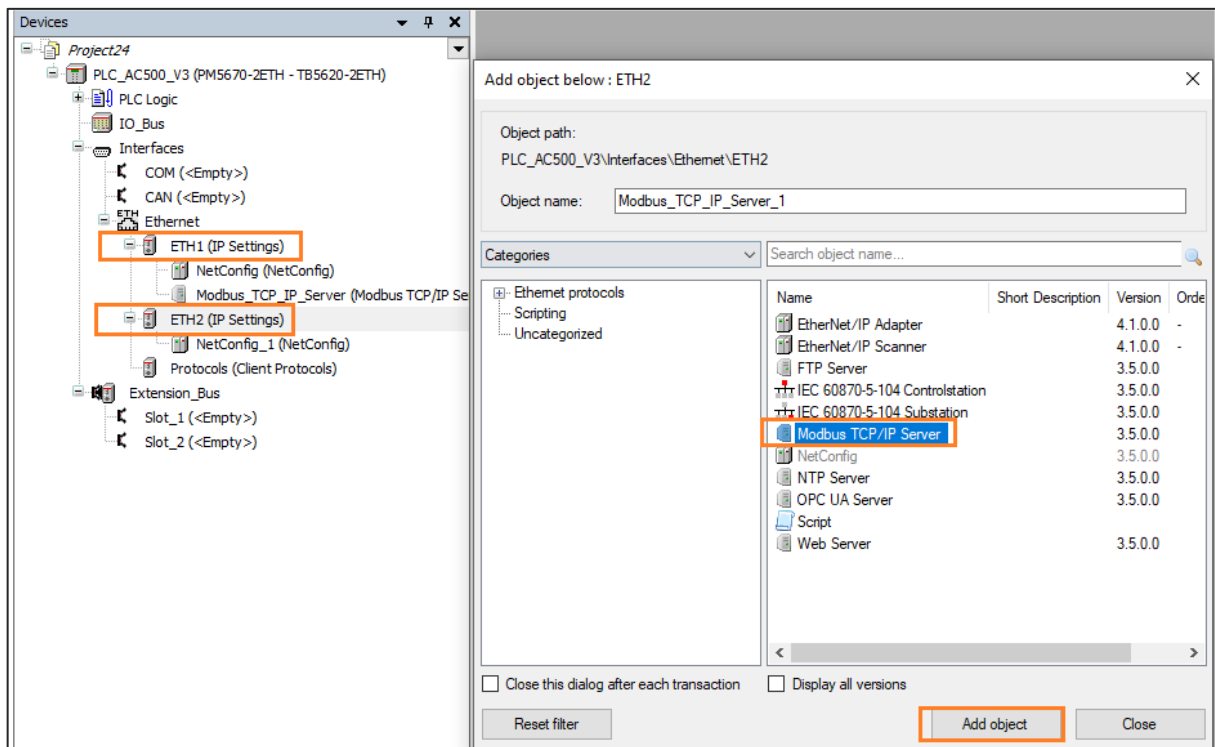
For detailed information on Modbus TCP communication please refer to the Automation Builder online help file.

Note: Multiple protocols can be added in parallel.

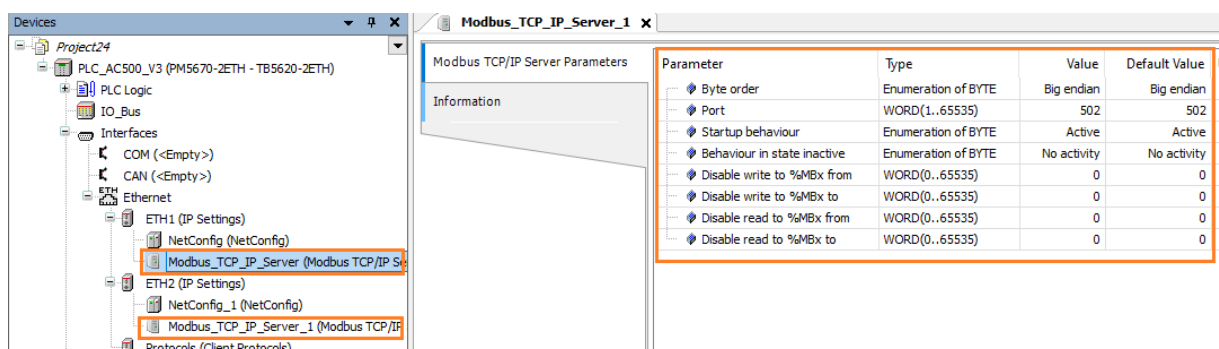
5.3.1 Modbus TCP/IP Server

In this Sending and receiving Modbus telegrams is performed automatically.

- A Modbus TCP/IP Server instance can be added to any specific Ethernet interface / IP address. Each interface supports max. one instance of “Modbus TCP/IP Server”. Right click on ETH interface and click “Add object”.
- The window “Add object below: ETH” appears. Select “Modbus TCP/IP Server” and click “Add object”.
- The node “Modbus_TCP_IP_Server” is added.



Server specific parameters to be set selecting the protocol's node “Modbus_TCP_IP_Server” under the respective ETH1 or ETH2.



Byte Order: Format/Endianness for the transmission of WORD values (register) within the request/response telegram (default: “Big Endian”).

Port:

TCP Port on which the Server listens.

Startup Behavior:

This parameter specifies how the Server behaves when configuration data is loaded (e.g. on download). It's default value is “Active”. This means the Server is immediately addressable after configuration has been performed. In case the Server should be activated later on during runtime by means of Function

Block “ModTcpServOnOff” this parameter value has to be set to “No activity”. Parameter behavior in state inactive then specifies the Server's behavior during the inactive phase.

Behavior in state “inactive”:

This parameter specifies how the Server behaves in an inactive state. This state may be set at the very beginning (parameter Startup Behavior = “No activity”) and/or requested during runtime calling Function Block “ModTcpServOnOff”. It's default value is “No activity”. This means the Server is not addressable at all (no listening socket on TCP/IP) when it is inactive. Using this setting, any requests by Modbus TCP Clients lead to the result Failed to connect to Server or Timeout. All other parameter values make the Server respond with an exception code to any requests by Modbus TCP Clients.

Disable Registers Section

It is possible to disable read and/or write access to individual segments. Reading/writing is disabled beginning at the set start address and is valid up to the set end address (inclusive).

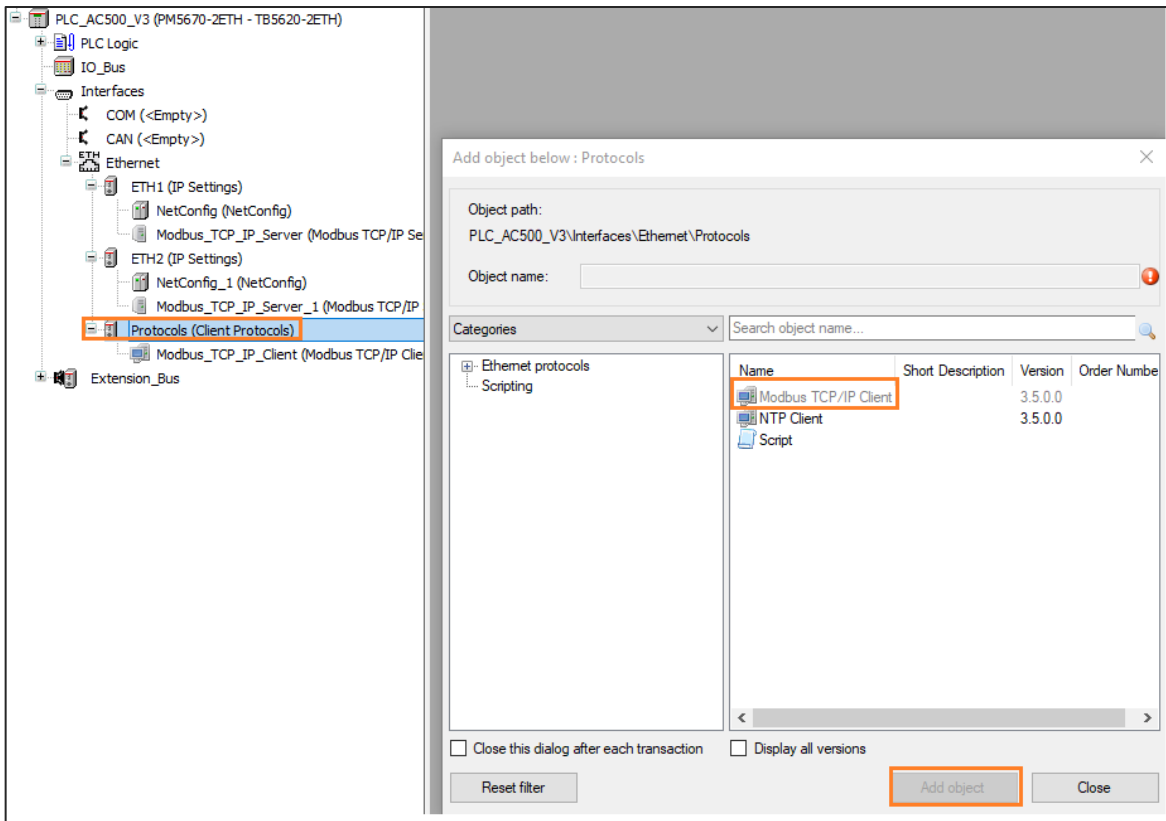
Parameter	Default	Value	Description
Disable write to %MB from	0	0 ... 65535	Disable write access starting at %MBx
Disable write to %MB to	0	0 ... 65535	Disable write access up to %MBx
Disable read from %MB from	0	0 ... 65535	Disable read access starting at %MBx
Disable read from %MBx to	0	0 ... 65535	Disable read access up at %MBx

5.3.2 Modbus TCP/IP Client

In this operating mode, the telegram traffic with the server(s) is handled via the Function Block. This Function Block sends Modbus request telegrams to the server(s) via the set interface and receives Modbus response telegrams from the server(s) via this interface.

The “Modbus_TCP_IP_Client” instance has to be added to the common Ethernet Client protocols' node. This node supports max. one instance of Modbus TCP/IP Client. Other protocols can be added in parallel.

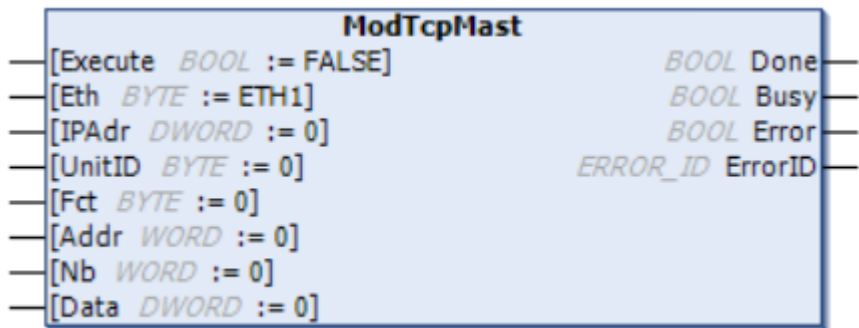
- Right click on the node “Protocols” and click “Add object”.
- The window “Add object below: Protocols” appears.
- Select “Modbus TCP/IP Client” and click “Add object”
- The node “Modbus_TCP_IP_Client” is added.



Depending on a Server's IP-Address the Client sends its requests via the Ethernet interfaces available.

Modbus TCP/IP Client does not have any parameters.

The Modbus Client functionality then must be realized with Function blocks (such as ModTcpMast) in the application program.

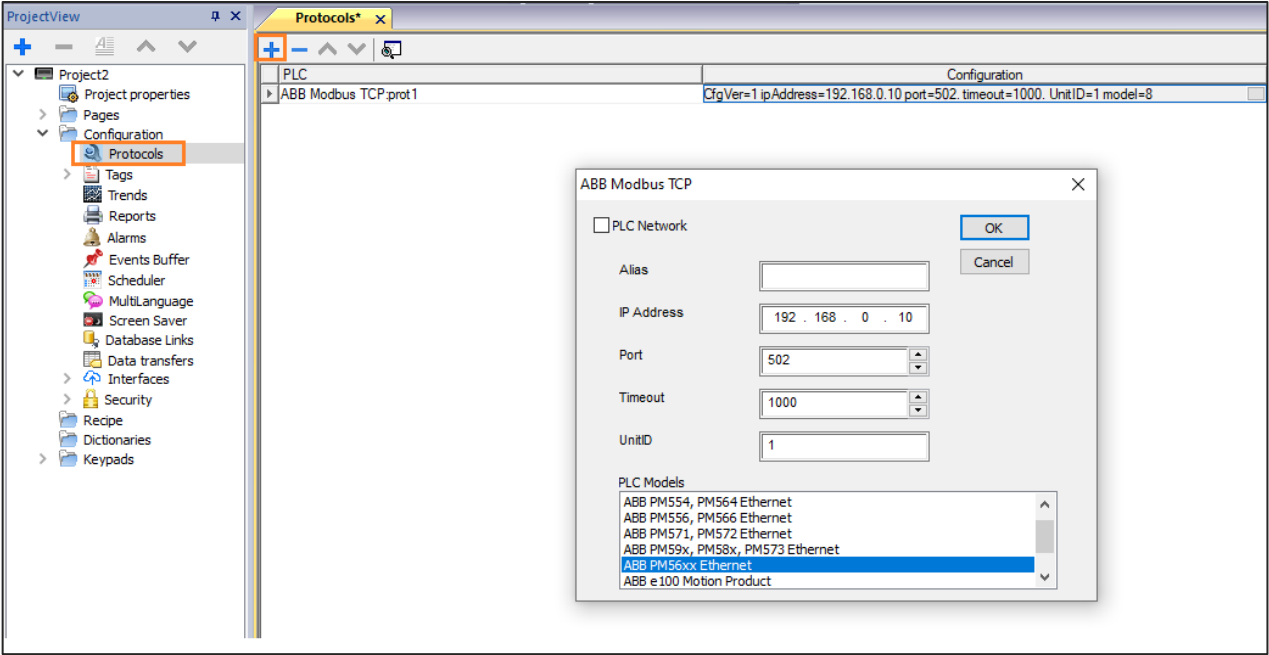


5.3.3 HMI's and Modbus TCP/IP communication

5.3.3.1 CP600 HMI Modbus TCP/IP communication

For CP600 HMI configuration, user must have installed Panel Builder software using Automation Builder Installation manager.

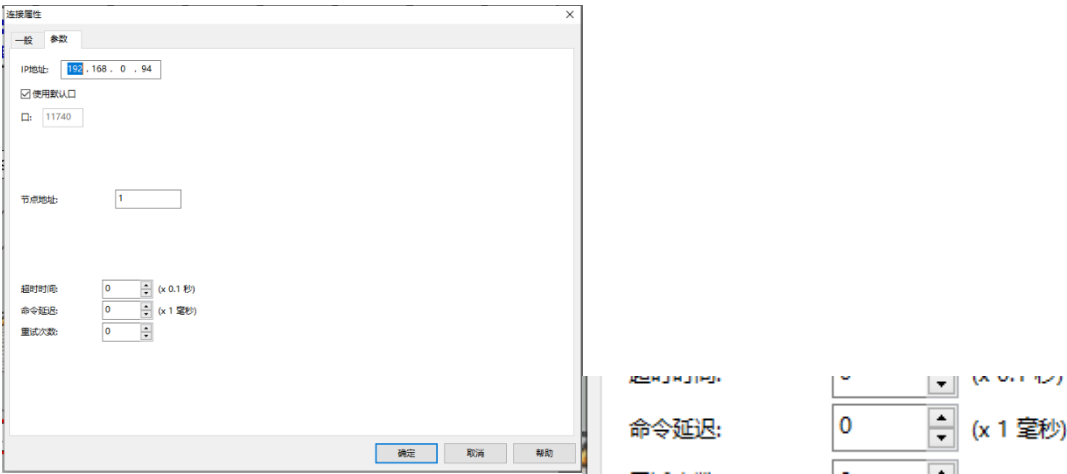
For CP600 communication with AC500 PLC, the internal AC500 / Codesys protocol is recommended. If the CP600 HMI should communicate via Modbus the user needs to configure AC500 ETH port as a ModbusTCP Server and the respective protocol selection and configuration needs to be done on the Panel builder software to establish the communication successfully.



More details on the CP600 configuration and communication with the AC500, please refer to the latest panel builder help file.

5.3.3.2 3rd Party Modbus TCP/IP HMI communication

If the user is using a 3rd Party HMI, its advised to take great care when using CoDeSys protocols. These 3rd party protocols sometimes have no data flow control and can poll the PLC so much that they can over-load the communication line. If using this protocol is unavoidable then check the settings for things such as ‘Command delay’ which can be used to add delays between Comms events.



5.4 OPC UA

OPC UA server can be added as an object below the Ethernet interfaces ETH1 or ETH2.The user can access the variable interface of the PLC via a client. At the same time, communication can be protected by means of encryption.

The OPC UA server supports the following features:

- Browsing of data types and variables
- Standard read/write services
- Notification for value changes: subscription and monitored item services
- Encrypted communication according to "OPC UA standard (profile: Basic256SHA256)"

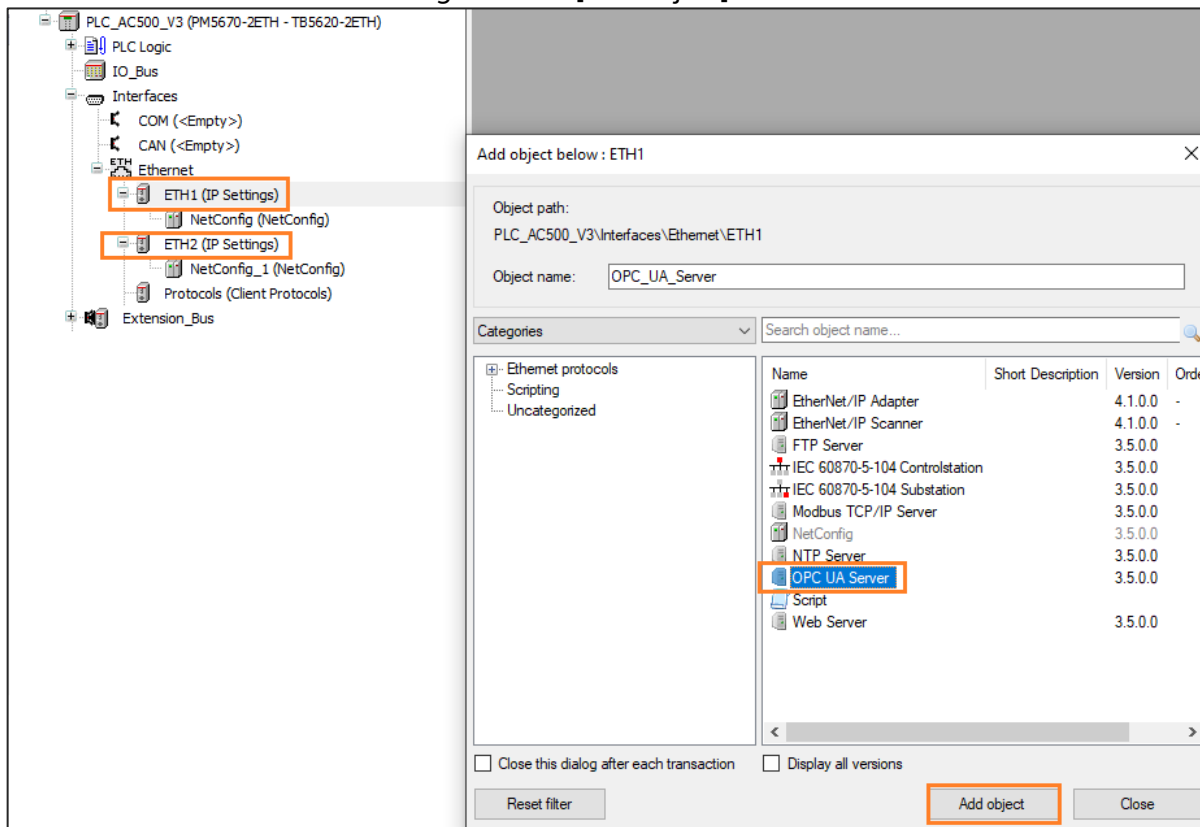
- Imaging of the IEC application according to "OPC UA Information Model for IEC 61131-3"
- Supported profile: Micro Embedded Device Server Profile
- By default, there is no restriction in the number of sessions, monitored items, and subscriptions. The number depends on the performance of the respective platform.
- Sending of events according to the OPC UA standard.

To add OPC UA to your application user needs to add the OPC UA Server and create the symbol configuration for the same. Follow the steps below to add the OPC UA into your application.

An Application Example is available to gain a deeper understanding of the OPC UA protocol and to configure AC500 V3 accordingly [How to use OPC Server V3 - for DA and UA](#)

Right-click on node ETH1 or ETH2 in your Automation Builder project and "Add object".

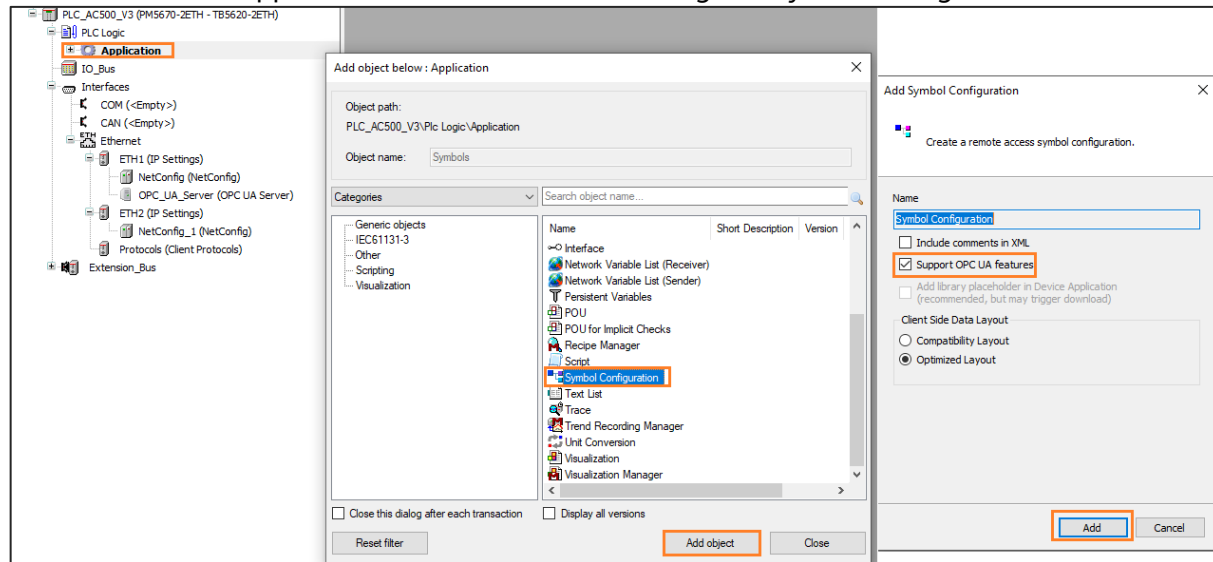
Choose OPC UA Server in the dialog and click [Add object].



Declare some variables of different types in the program.

Right-click "Application" then click "Add object". Choose Symbol configuration and click [Add object].

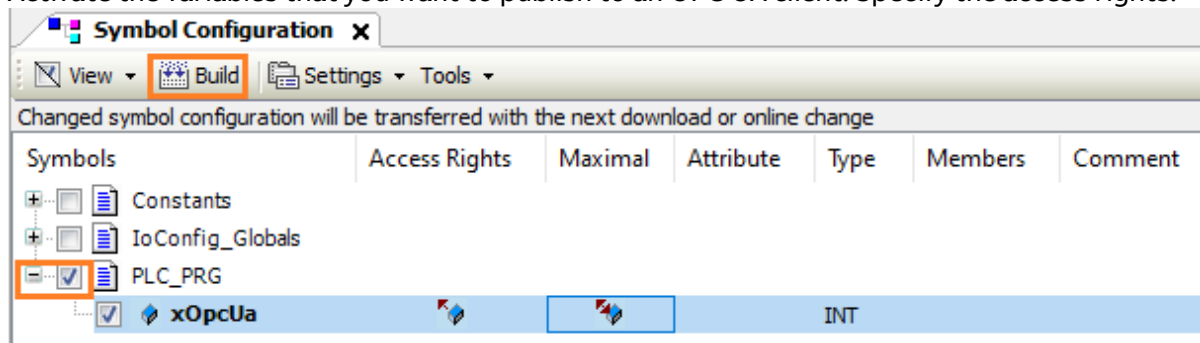
Enable checkbox Support OPC UA Features in the dialog Add symbol configuration.



Double-click “Symbol configuration” in the Devices tree to open the editor Symbol configuration.
Click [Build].

The variables are displayed in a tree structure.

Activate the variables that you want to publish to an OPC UA client. Specify the access rights.



Download the project to the PLC.

Now use a OPC UA Client and browse for the OPC UA Server PLC to access the variables.

For more details on OPC UA configuration and other functionality please refer to the latest Automation Builder online help file.

6 GETTING ONLINE AND MANAGING THE PLC

6.1 Getting online to the PLC

6.1.1 Set-up communication parameters in windows

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

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

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

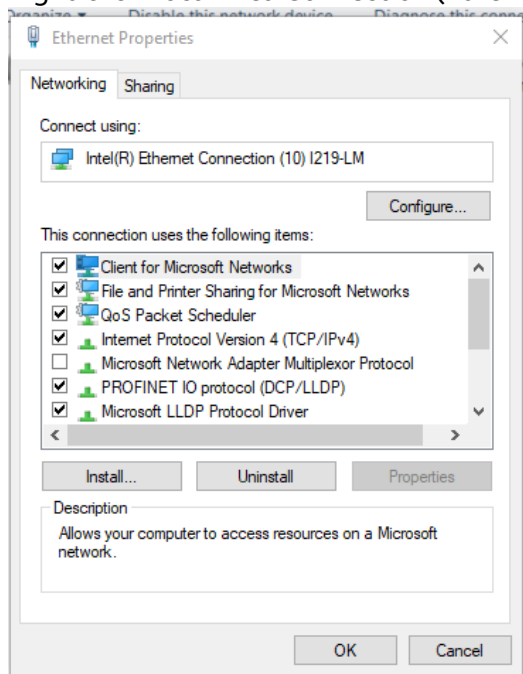
The IP address of your PC should be 192.168.0.X. Avoid X = 10 to prevent an IP conflict with the CPU. Subnet mask should be 255.255.255.0.

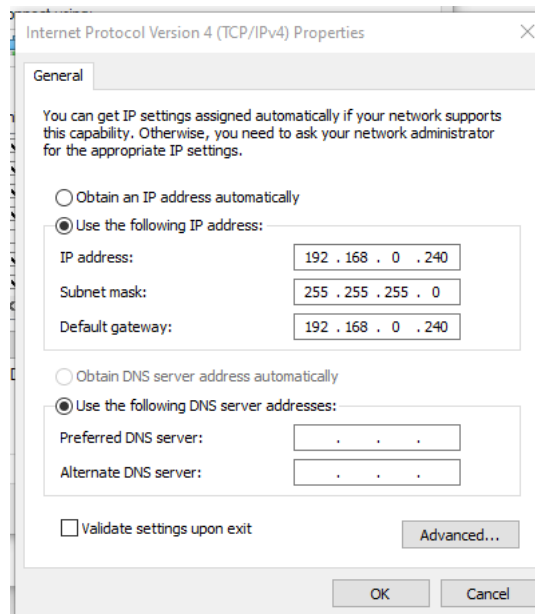
6.1.1.1 Change the windows IP address

Open Windows Control Panel. Click "Ethernet Settings".

Click Change adapter Options.

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





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

Enter your desired IP address and subnet mask.

6.1.2 Configuration of the PLC IP settings

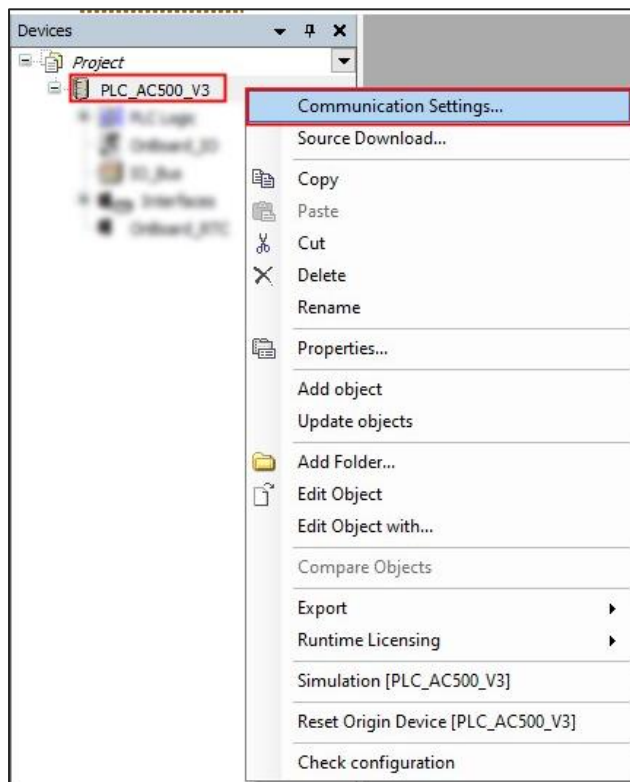
The factory setting of the IP address of the CPU is 192.168.0.10 and subnet mask is 255.255.255.0. The user can change the same using IP Configuration tool.

6.1.3 Set-up the communication gateway

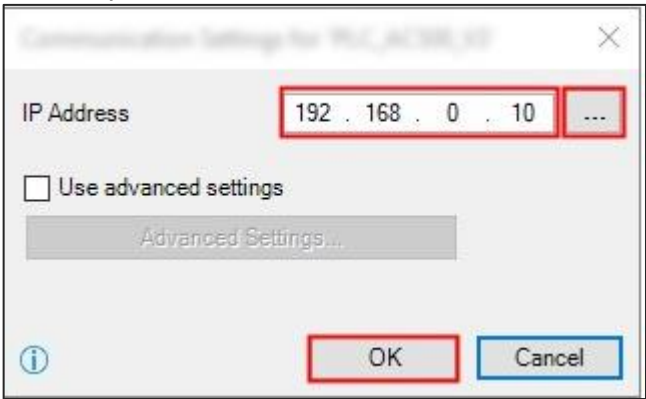
Make sure the CPU and computer are connected with an Ethernet cable.

6.1.3.1 PLC IP Address is known to the user

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



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



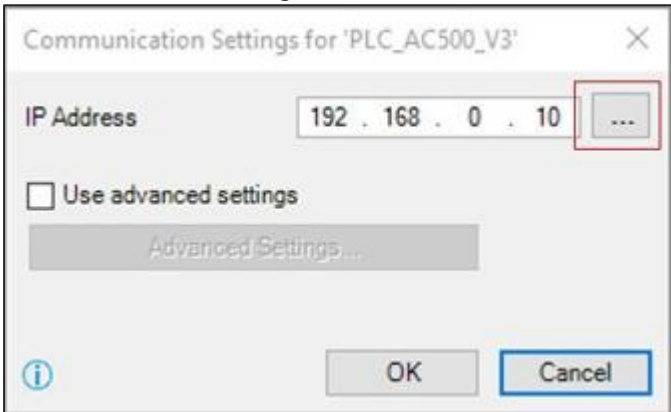
- Select “OK” to implement the IP address

6.1.3.2 PLC IP Address is not known to the user (Network scan)

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

Right-click “PLC_AC500_V3” in the device tree.

Select “Communication Settings”.

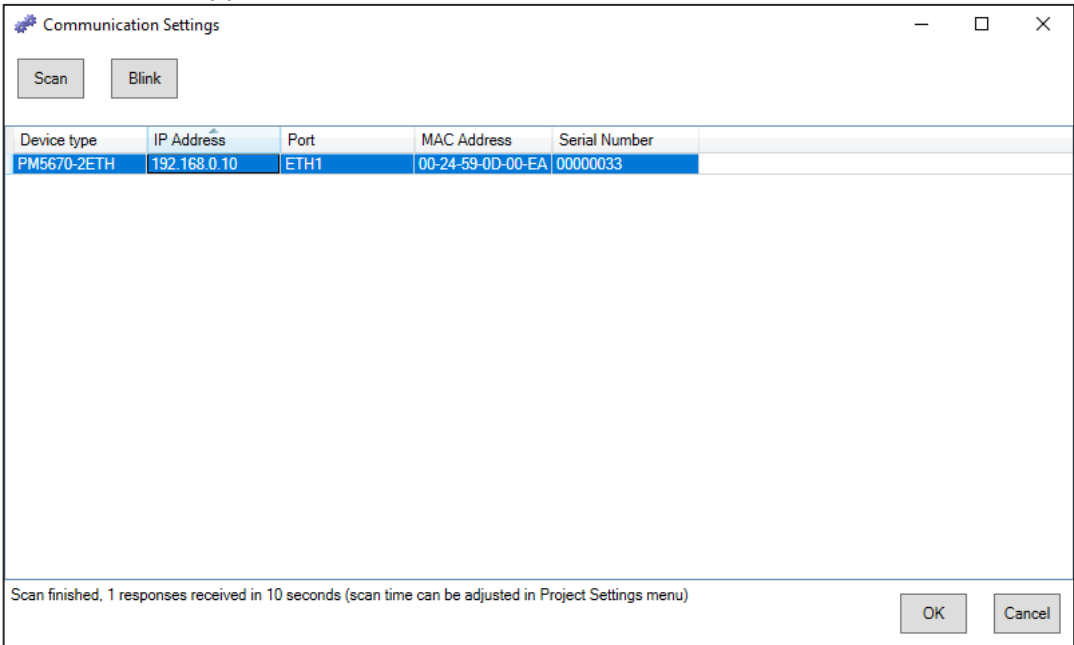


Select “...”.

Pick IP Address for "PLC_AC500_V3" opens.

The automatic scan runs.

The results will appear in this field.



Select the CPU in the field and select “OK” to implement the needed communications gateway.

6.1.4 Check communication settings

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

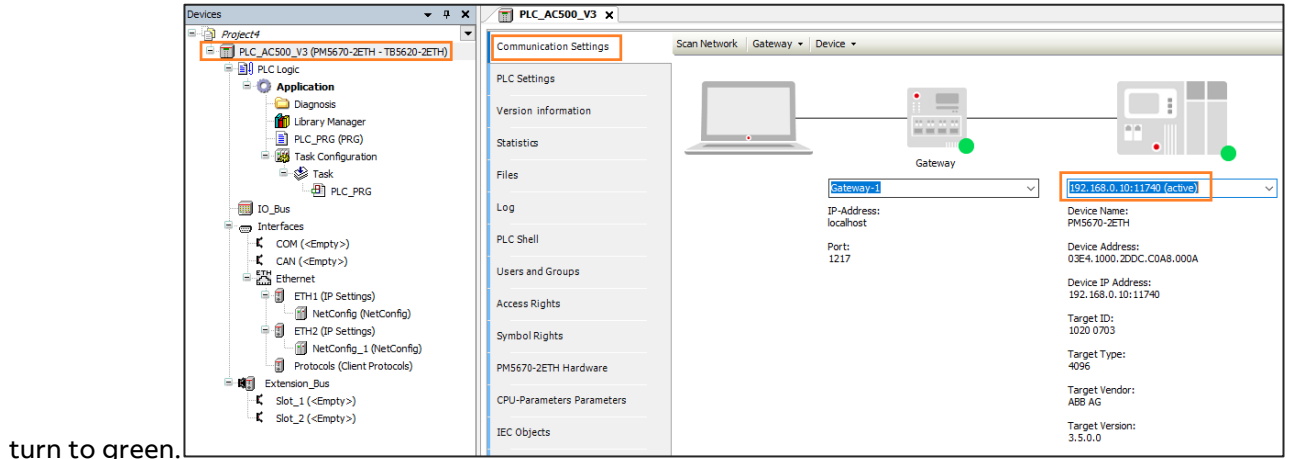
Double-click “PLC_AC500_V3” in the device tree.

Select “Communication Settings”.

The selected IP address is shown.

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

To test the connection and/or to see the CPU information press [Enter] or click on the black dot next to the PLC picture. If the connection is proper, PLC information will be shown below and the black dot will



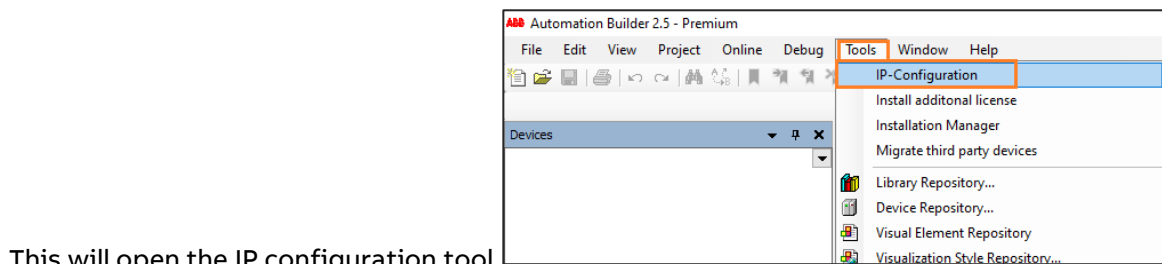
turn to green.

6.1.5 Change PLC IP address

User can change the PLC IP address by using Automation Builder or by using the function keys and display on the processor module. For more details on how to use the function keys and display to read the current IP address or to change the IP address, please refer the latest Automation Builder help file.

6.1.5.1 IP configuration using Automation Builder / IP Configuration tool

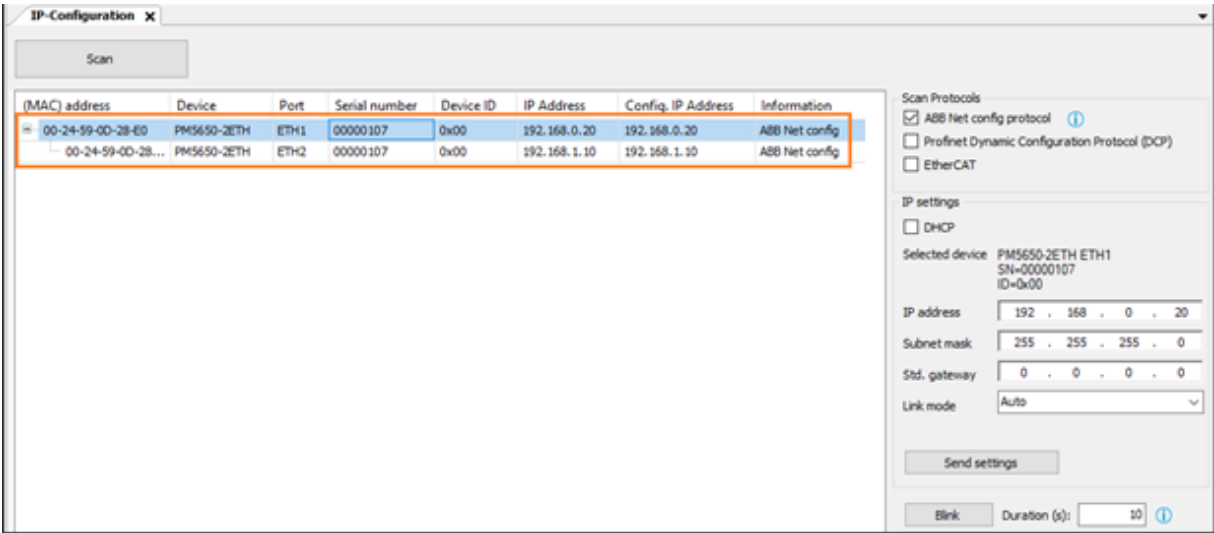
User can launch the integrated IP configuration tool from Automation Builder by clicking on menu “Tools -> IP-Configuration”.



This will open the IP configuration tool

Click on the “Scan” button to start scanning the devices.

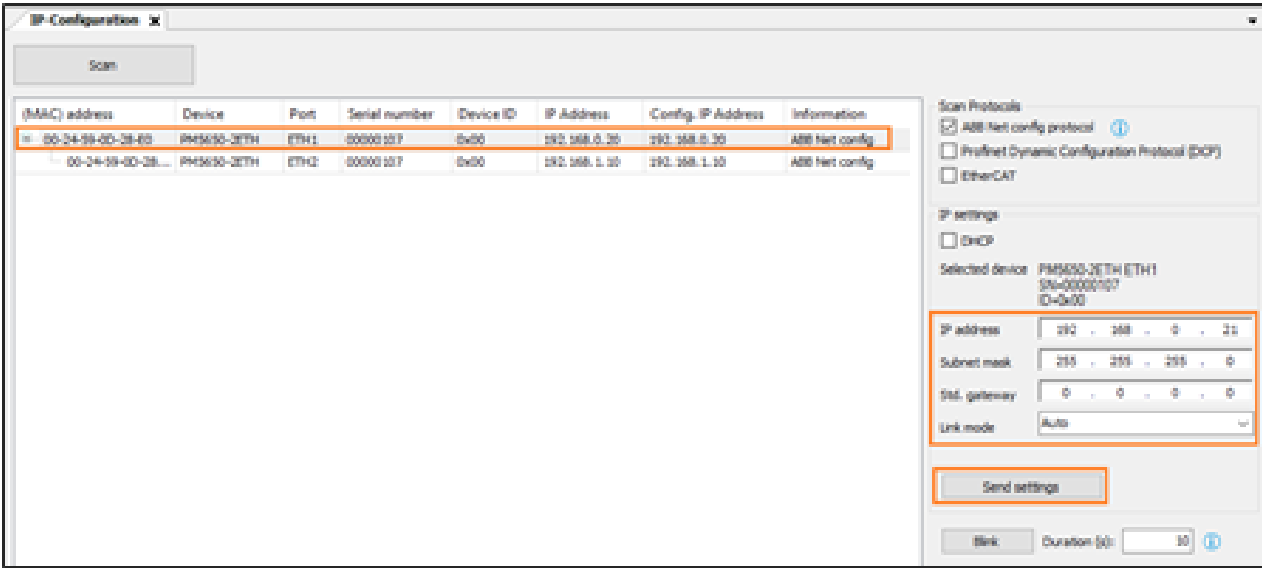
After finishing the scanning, all the connected devices will be listed with the Device name and Ip address.



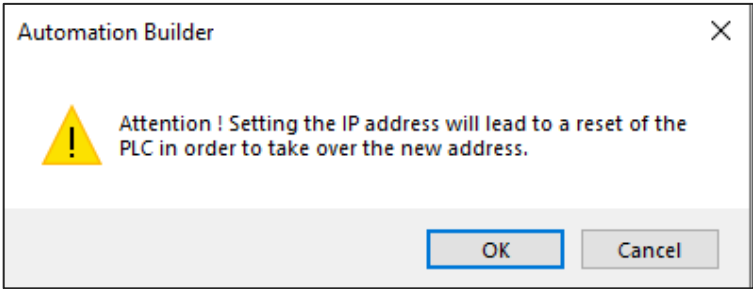
Click on the ETH port to change the IP address and enter the new IP address.



Note: Make sure the PLC is in STOP mode. Check the Run led is not glowing / LED display is showing STOP.
If the PLC is in RUN mode, press the function key “run” once to STOP the PLC.



Click on “Send setting” to send the new IP address to PLC and a warning message will be appeared as below since after setting the new IP address the PLC will be restarted once automatically.



After sending the IP address successfully a “Send succeeded” will be appeared on the bottom.

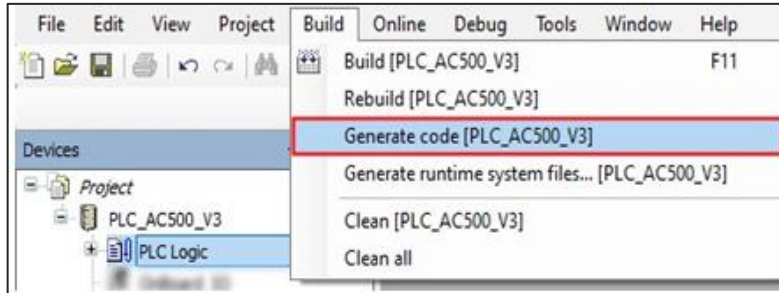
After restarting the PLC automatically user can “scan” the devices using “IP configuration” and find the ETH port with new IP address.

For more details on IP configuration tool, please refer the Automation Builder help file.

6.2 Login to the CPU and download the program

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

Before logging-in to the CPU, you need to compile the complete code without any errors. Select menu “Build -> Generate code”. The result of the compilation is shown in the “Messages” field at the bottom of the screen.

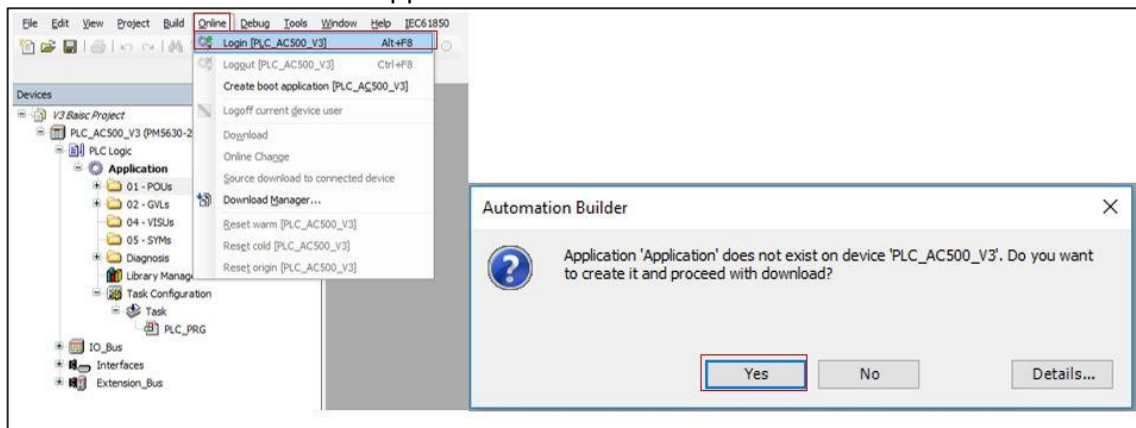


If you skip the compiling and select “Login”, the Automation Builder will automatically trigger compiling in advance to logging-in.

In the Automation Builder menu select “Online -> Login [PLC_AC500_V3]”.

A pop-up will appear

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



PLC in STOP mode STOP Program loaded

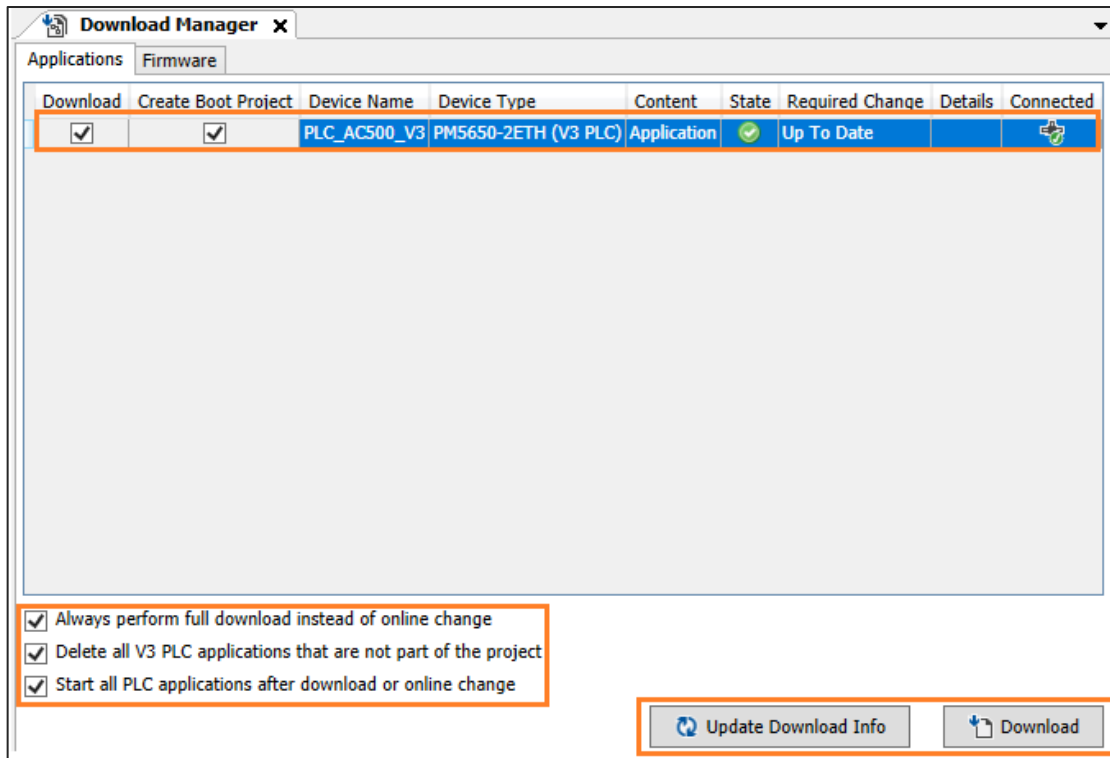
Menu select “Debug -> Start [PLC_AC500_V3]” to RUN the PLC RUN Program loaded

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

The project is stored in the RAM memory. The project is stored in the flash EEPROM, if boot application was created. Select menu “Online -> Create Boot Application”.

Alternatively, the user can also use the Download Manager from menu “Online -> Download manager”. Check the options needed by the user and click on the Download button.

User can also use Download manager for downloading multiple projects to different PLC's.

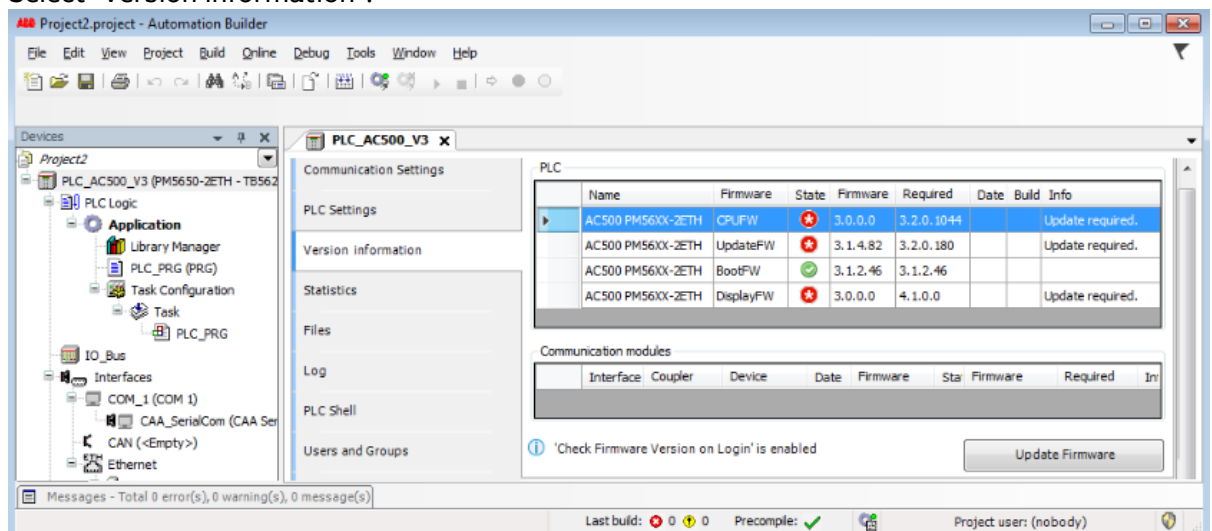


6.3 Firmware update

The PLC firmware can be updated via Automation Builder.

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


- An Automation Builder project with an AC500 V3 CPU is open.
- CPU is in "stop" mode or shows **"uPdAtE"** (update) on the display.
- After an update, the CPU shows either **"donE"** or **"StoP"** on the display
- For new modules: IP address is set. (The default IP address is 192.168.0.10)
- Double-click CPU "PLC_AC500_V3".
- Select "Version information".



- Select "Update Firmware". While the update process is running, the RUN and ERR LEDs are toggling, i.e., they are flashing alternating.

- Wait for the PLC to finish the update.

A completed update is indicated by a message on the display. Either “doneE”, or “StoP”.



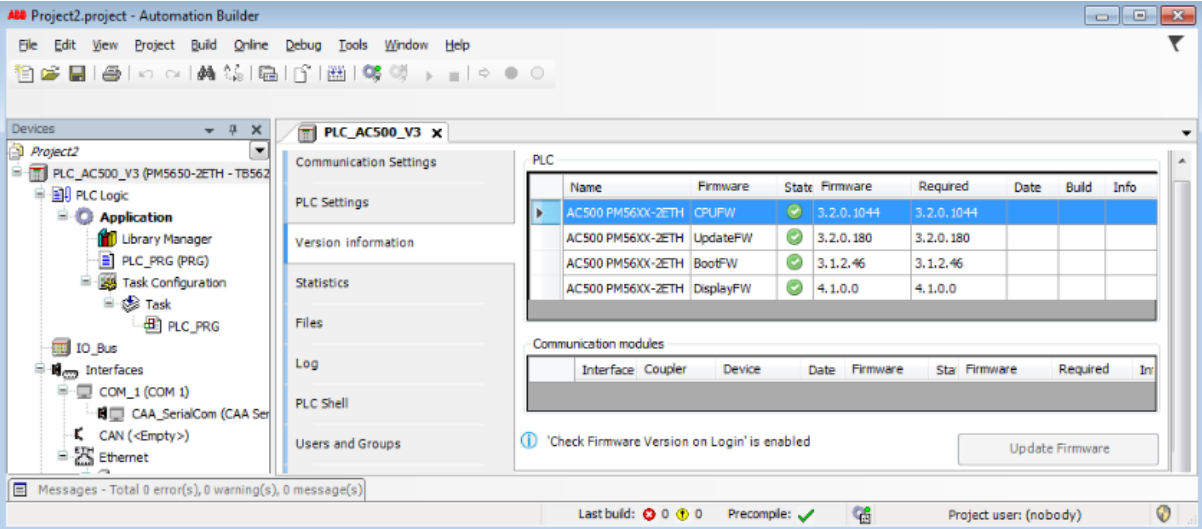
Note: Do not disconnect the power supply during the update process! The PLC could be damaged.

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

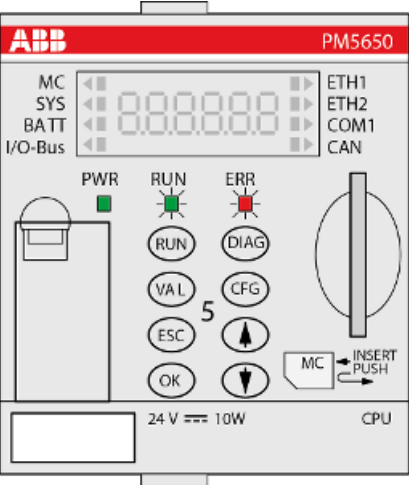
The CPU display shows "stop" after re-boot. The update process is finished.

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

Successful firmware update:



6.3.1 Behaviour of LEDs during firmware update



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

6.4 Run time license for PLC for Motion Control

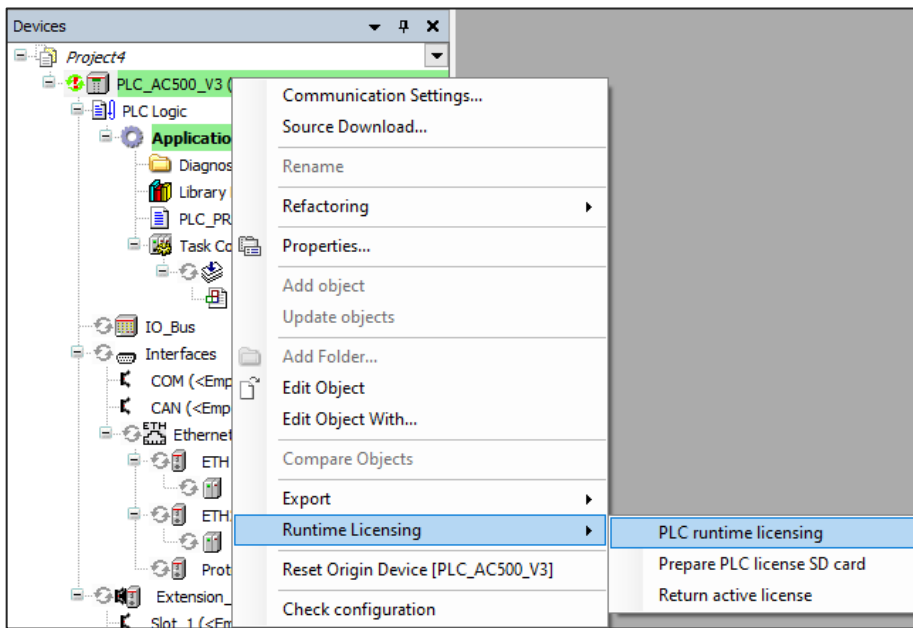
6.4.1 What is Run time licensing

The use of some libraries, CPU features and devices requires the PLC to have a CPU locked runtime license activated inside the CPU.

There are many ways to activate and remove the run time license from CPU. User can activate or remove the license from CPU when it is online or offline with a computer or when the computer is having an internet connection or not. Some of the commonly used CPU run time license activation or license removal method is explained in below chapters. For more options on license activation, refer to the Automation Builder help file.

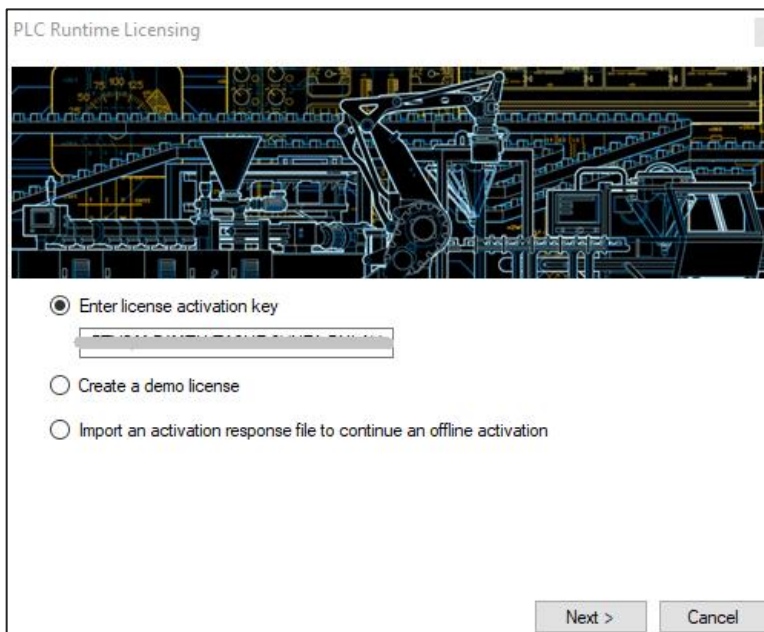
6.4.2 Activating PLC license with internet connection

Right-click on the PLC and select “PLC runtime licensing” from the “Runtime Licensing” menu.

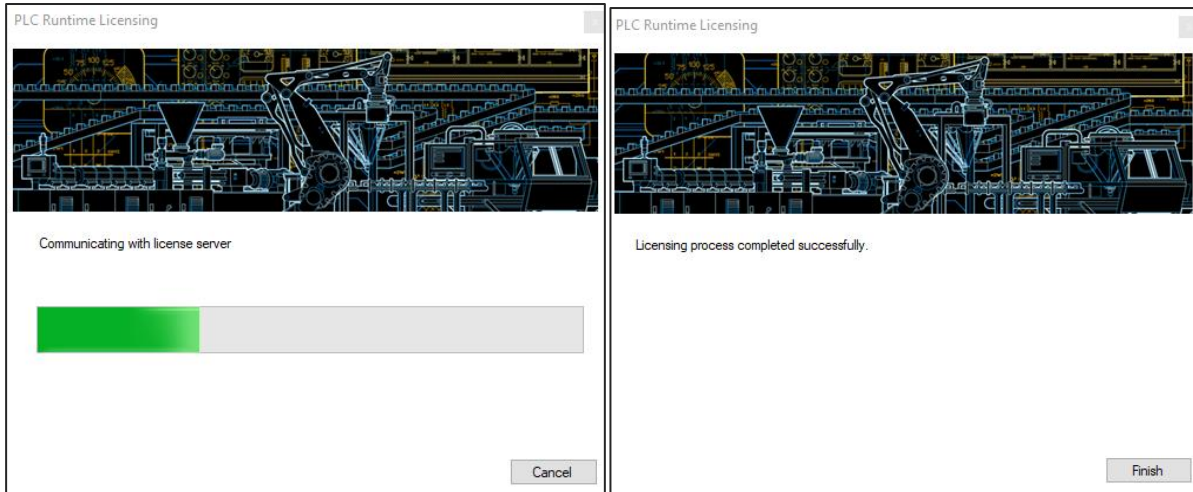


A wizard starts. Follow the instructions.

Enter the license activation key and select “Next” to finish the licensing procedure.

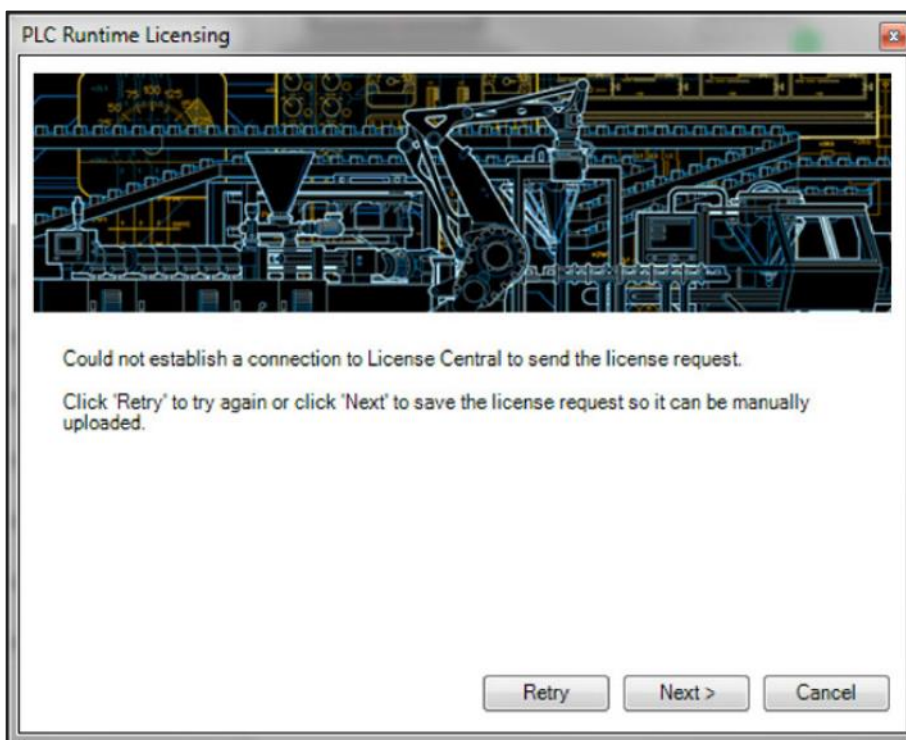


The license is activated on the PLC device.



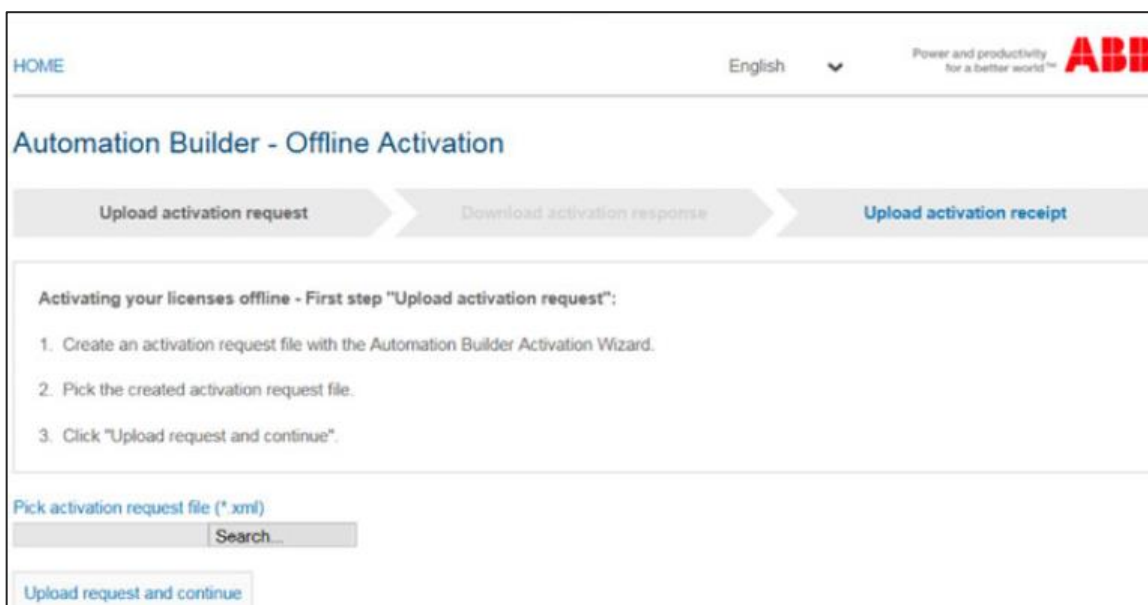
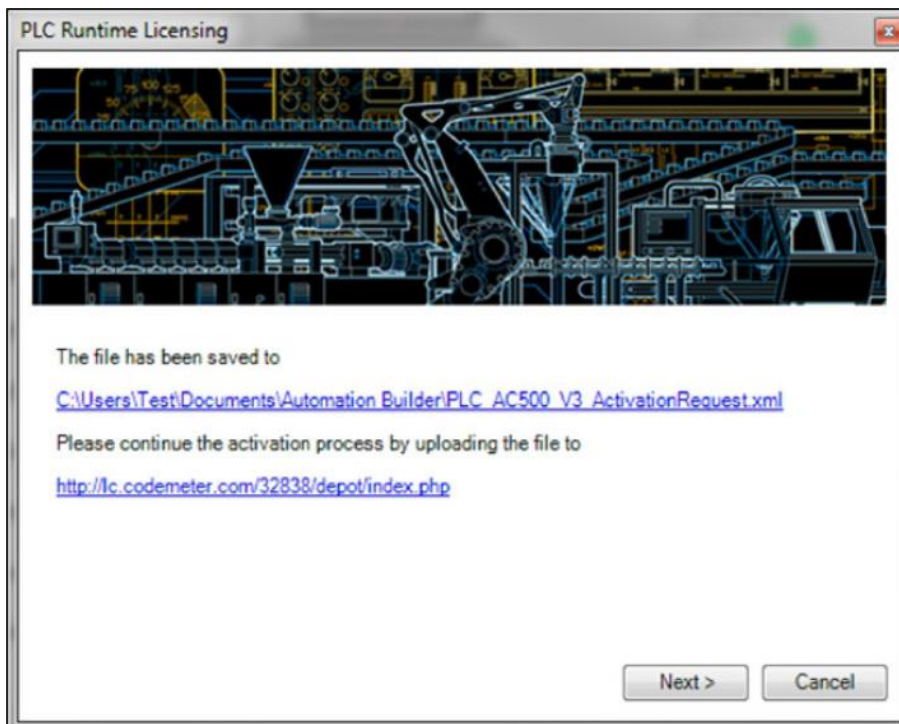
6.4.3 Downloading and activating PLC license without internet connection

If an error occurs when communicating with the ABB license server, or if Automation Builder is running on a computer without internet connection, then it is possible to manually complete the ABB license server interaction by using another computer (with internet connection).



In the error dialog select “Next” and save the license activation request file to a storage location the other computer can access, e.g. a file share.

In the dialog the web address of the ABB license server is displayed (<http://lc.codemeter.com/32838/depot/index.php>). From the computer with internet connection, upload the license activation request file.



After the upload, download and save the license activation file from the ABB license server. Transfer this file to the PC without Internet connection.

Select “Next” to continue the license activation process. Click “Cancel” to continue the license activation process later.

Select “Browse” and select the license activation file (*.WibuCmRaU) from the defined storage location.

The license is validated by the ABB license server and afterwards activated on the PLC device.

If the license shall be used on another PLC device, the installed license can be returned.

To complete the licensing process, a license receipt file must be uploaded to the ABB license server.

Save the license receipt file and upload it manually from a PC with internet connection to <http://lc.codemeter.com/32838/depot/index.php>.

A license confirmation is returned.

6.4.4 Downloading and activating PLC license via memory card

When you have no connection between your PC and the PLC device the licensing procedure can be done via a memory card.



Note: There is a connection to the internet.

A memory card which can be used with AC500 V3 products.

On the PC: Create a license request

Place the memory card in the PC.

Right-click on the PLC node and select “Prepare PLC license SD memory card” from the “Runtime Licensing” menu.

From the filesystem select the root folder of the memory card.

A success message is displayed when the creation of the memory card files is completed.

The license request files are stored to the selected folder.

On the PLC: Transfer the license data

Insert the memory card into the PLC device and reboot the PLC.

When the license request file is successfully created by the PLC, “done” is shown on the display of the PLC.

Remove the memory card from the PLC.

On the PC: Enter the license activation key

1. Place the memory card into the PC.
2. Open the PLC project in Automation Builder. Ensure the PLC is logged out.
3. Right-click on the PLC node and select “PLC runtime licensing” from the “Runtime Licensing” menu.
 - a. A wizard is started. Follow the instructions.
4. Enter the license activation key.
5. From the filesystem, select the root folder of the memory card.

- a. The previously created license request files are sent to the ABB license server. A license activation is created on the memory card.

6. Remove the memory card from the PC.

On the PLC: Complete license activation for the PLC

Insert the memory card into the PLC device and reboot the PLC.

done is displayed on the PLC if license activation was successful.

Remove the memory card from the PLC

On the PC: Complete license activation on the license server

To complete the licensing process, the license receipt file must be uploaded to the ABB license server.

1. Place the memory card into the PC.
2. Upload the license receipt file manually from a PC with internet connection to

<http://lc.codemeter.com/32838/depot/index.php>.

A license confirmation is returned.

6.4.5 Activating a demo license

It is possible to try out device features or library features by using a Demo license on the PLC. With this, you can use the features for a limited time.

Right-click on the PLC node and select “PLC runtime licensing” from the “Runtime Licensing” menu. A wizard is started. Follow the instructions.

Select the option “Create a demo license” and click “Next” to finish the licensing procedure. The demo license is validated by the ABB license server and afterwards activated on the PLC device.

The demo license is valid for all licensed features and the duration of the demo license is 10 days PLC in run. After expiration, further demo licenses can be activated. As it is a demo license, there is no grace period after expiration (product licenses do not have an expiration date)

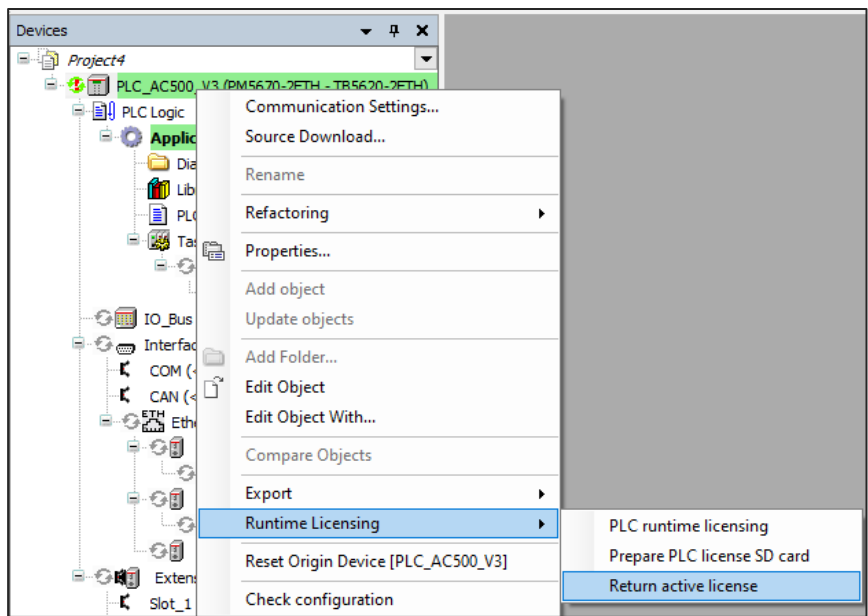
For standard AC500 V3 it is visible on the display, if the PLC is running on demo license

6.4.6 Returning a license from a PLC

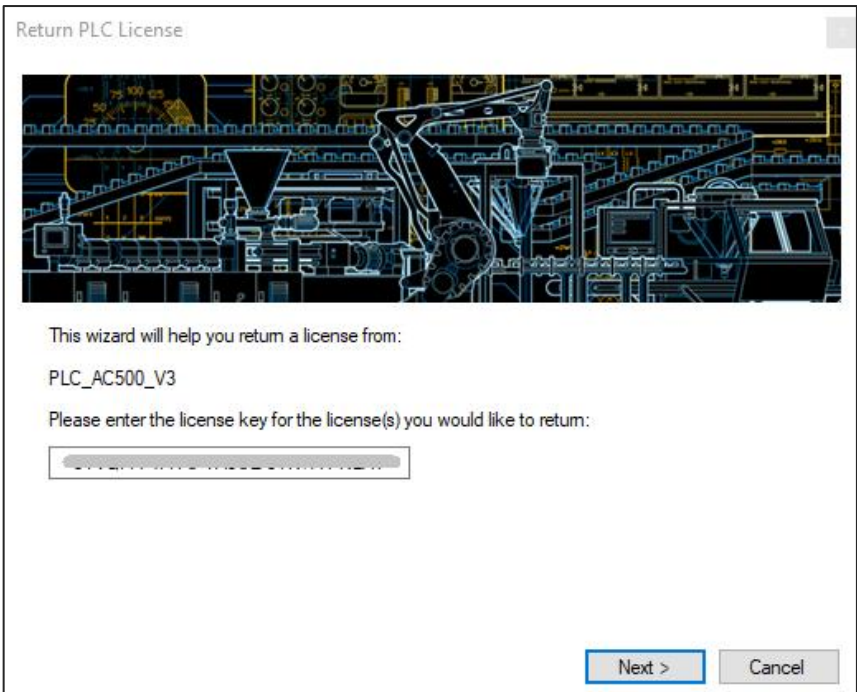
A license which has been installed on a PLC can be returned and installed on another PLC.

6.4.6.1 Returning a license using Automation Builder

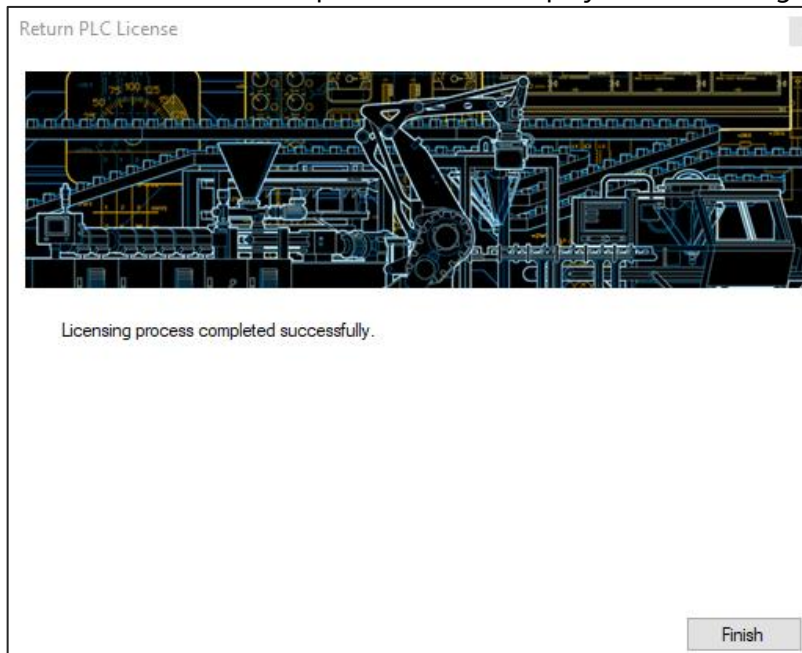
1. Right-click on the PLC node and select “Return active license” from the “Runtime Licensing” menu. A wizard is started. Follow the instructions.



2. Enter the license activation key and click “Return license”.



The results of the return process will be displayed in the dialog.



The license from the PLC device is removed/returned and can be used now for another PLC.

6.4.6.2 Returning a license using SD card

When the PLC is not connected to the PC (PLC logged out) it is possible to return a license via memory card.

1. Place the memory card into the PC.
2. Right-click on the PLC node and select "Return active license" from the "Runtime Licensing" menu.

A wizard is started. Follow the instructions.

3. Enter the license activation key and click "Return license".
4. Click "Browse" and select the root folder of the memory card.

Returning of the license is started.

5. Place the memory card in the PLC device and reboot the PLC.

The license from the PLC device is removed and can be used now for another PLC device.

6. To complete the licensing process, a license receipt file must be uploaded to the ABB license server.
7. Save the license receipt file and upload it manually from a PC with internet connection to <http://lc.codemeter.com/32838/depot/index.php>.

A license confirmation is returned

7 GENERAL PLC PROGRAM BASICS

This chapter is covering limited program basics and program editor features which is supported by AC500 PLC. User can refer to latest Automation Builder helpfile chapter “Programming with IEC 61131-3 editor” for detailed information on programming AC500 PLC’s and all supported functionalities while programming AC500 PLC.

7.1 Programming languages and editors

Automation Builder offers a text editor for ST and graphic editors for SFC, FBD/LD/IL and CFC. You can program a POU in each case in the editor for the implementation language that you selected when creating the POU. The editor opens with a double-click on the POU in the device tree or in the POUs view.

Each of the programming language editors consists of two sub-windows:

In the upper part you make declarations in the “declaration editor”, in text or tabular form depending on the setting. In the lower part you insert the implementation code in the respective language.

You can configure the display and the behavior of each editor project-wide on the associated tab of the options.

7.2 Variable classifications

The scope of a variable defines how and where you can use a variable. You define the scope in the variable declaration.

7.2.1 Local Variables - VAR

Local variables are declared between the keywords VAR and END_VAR in the declaration part of programming objects.

You have read-only access to local variables by using the instance path.

You can extend local variables with an attribute keyword.

Example

```
VAR
    iVar1 : INT;
END_VAR
```

7.2.2 Input Variables - VAR_INPUT

Input variables are used at the inputs of function blocks. VAR_INPUT variables are declared between the keywords VAR_INPUT and END_VAR in the declaration part of programming objects.

You can extend input variables with an attribute keyword.

Example

```
VAR_INPUT
    iIn1 : INT; (* 1st input variable *)
END_VAR
```

7.2.3 Output Variables - VAR_OUTPUT

Output variables are used at the outputs of function blocks.

VAR_OUTPUT variables are declared between the keywords VAR_OUTPUT and END_VAR in the declaration part of programming objects. CODESYS returns the values of this variable to the calling POU. There you can retrieve the values and continue using them.

You can extend output variables with an attribute keyword.

Example

```

VAR_OUPUT
    iOut1 : INT; (*1st output variable *)
END_VAR

```

7.2.4 Input/Output Variable (VAR_IN_OUT)

A VAR_IN_OUT variable is an input/output variable, which is part of a POU interface and serves as a formal pass-by-reference parameter.

Syntax declaration

<keyword> <POU name>

```

VAR_IN_OUT
    <variable name> : <data type> ( := <initialization value> )? ;
END_VAR
    <keyword> : FUNCTION | FUNCTION_BLOCK | METHOD | PRG

```

You can declare an input/output variable in the VAR_IN_OUT declaration section in the POUs PRG, FUNCTION_BLOCK, METHOD, or FUNCTION. As an option, a constant of the declared data type can be assigned as an initialization value. The VAR_IN_OUT variable can be read and written.

7.2.5 Global Variables - VAR_GLOBAL

Global variables are ordinary variables, constants, external or remanent variables that are recognized within the entire project.

You declare global variables in global variable lists or in the declaration section of programming objects between the keywords VAR_GLOBAL and END_VAR.

The system recognizes a global variable when you prepend the variable name with a dot (for example, .iGlobVar1).

Example

```

VAR_GLOBAL
    iVarGlob1 : INT;
END_VAR

```

7.2.6 Temporary Variable - VAR_TEMP

This function is an extension of the IEC 61131-3 standard.

You declare temporary variables locally between the keywords VAR_TEMP and END_VAR.

VAR_TEMP declarations are possible only in program blocks and function blocks.

CODESYS initializes temporary variables each time the block is called.

The application can access the temporary variables only in the implementation section of a program block or a function block.

Example

```

VAR_TEMP
    iVarTmp1 : INT; (*1st temporary variable *)
END_VAR

```

7.2.7 Static Variables - VAR_STAT

This function is an extension of the IEC 61131-3 standard.

You declare static variables locally between the keywords VAR_STAT and END_VAR. CODESYS initializes static variables the first time each block is called.

You can access static variables only from within the namespace where the variables are declared (like static variables in C). But static variables retain their values when the application leaves the block. For example, you can use static variables as counters for function calls.

You can extend static variables with an attribute keyword.

Example

VAR_STAT

iVarStat1 : INT;

END_VAR

7.2.8 Constant Variables - 'CONSTANT'

Constant variables are declared in global variable lists or in the declaration part of programming objects. In implementations, constant variables can be accessed as read-only via the instance path.

Always assign an initialization value when declaring a constant variable. Then the constant cannot be written any more.

Example

Declaration

VAR CONSTANT

c_rTAXFACTOR : REAL := 1.19;

END_VAR

Call

rPrice := rValue * c_rTAXFACTOR;

You have read-only access to constant variables in an implementation. Constant variables are located to the right of the assignment operator.

7.2.9 Persistent Variable - PERSISTENT

Persistent variables are declared in the declaration section VAR_GLOBAL RETAIN PERSISTENT in the persistent global variable list. For variables that are marked with the PERSISTENT keyword outside of the persistence editor, instance paths are added there.

Syntax of the declaration in the global persistent variable list PersistentVars:

VAR_GLOBAL PERSISTENT RETAIN

<identifier> : <data type> (:= <initialization>)?;

<instance path to POU variable>

END_VAR

Syntax of the declaration in POUs

<scope> PERSISTENT RETAIN

<identifier> : <data type> (:= <initialization>)?; // (...)? : Optional

END_VAR

<scope> : VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | VAR_STAT | VAR_GLOBAL

7.2.10 Retain Variable - RETAIN

Retain variables are declared by the keyword RETAIN is added in programming objects in the scope VAR, VAR_INPUT, VAR_OUTPUT, VAR_IN_OUT, VAR_STAT, or VAR_GLOBAL.

Syntax for the declaration

```
<scope> RETAIN
    <identifier>: <data type> (:= <initialization> )? // ( ... )? : Optional
END_VAR
<scope> : VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | VAR_STAT | VAR_GLOBAL
```

Example

In a POU:

```
VAR RETAIN
    iVarRetain: INT;
END_VAR
```

In a GVL:

```
VAR_GLOBAL RETAIN
    g_iVarRetain: INT;
END_VAR
```

7.2.11 Handling of remanent variables for AC500 V3 products

Maintaining data after power ON/OFF, is only possible, if a battery is connected for AC500 CPU and the buffering will take place in AC500 V3 CPU. The following data can be buffered completely or in parts:

- Data in the addressable flag area (%M area)
- RETAIN variable
- PERSISTENT variable (number is limited, no structured variables)
- PERSISTENT area

To prevent data loss when using the AC500 battery, the battery status should be periodically monitored by the user program.

The battery status can be monitored either with the help of a user program on the PLC or in Automation Builder.

7.3 Data types

7.3.1 BOOL

Data Type	Value	Memory
BOOL	TRUE (1), FALSE (0)	8 bit

7.3.2 INTEGER

CODESYS provides the following integer data types. Information can be lost when converting from larger to smaller types.

Data Type	Lower Limit	Upper Limit	Memory
BYTE	0	255	8 bit
WORD	0	65535	16 bit
DWORD	0	4294967295	32 bit
LWORD	0	$2^{64}-1$	64 bit
SINT	-128	127	8 bit
USINT	0	255	8 bit
INT	-32768	32767	16 bit
UINT	0	65535	16 bit
DINT	-2147483648	2147483647	32 bit
UDINT	0	4294967295	32 bit
LINT	-263	$2^{63}-1$	64 bit
ULINT	0	$2^{64}-1$	64 bit

7.3.3 REAL / LREAL

The data types REAL and LREAL are floating-point types according to IEEE 754. They are necessary when using decimal numbers and floating-point numbers in decimal notation or exponential notation.

Data type	Smallest value number	Largest value number	Storage space
REAL	1.00E-44	3.40E+38	32 bit
LREAL	4.94065645841247E-324	1.7976931348623157E+308	64 bit

7.3.4 STRING

A variable of data type STRING can have contain any character string. The amount of memory that is reserved during a declaration refers to characters and is shown in parentheses or brackets. If a size is not defined, then CODESYS allocates 80 characters by default.

As a rule, CODESYS does not limit the string length. However, the string function processes lengths of 1–255 only. If a variable is initialized with a string that is too long for the data type, then CODESYS truncates the string accordingly from the right.

The memory required for a STRING variable is always one byte per character plus one additional byte (for example, 81 bytes for a "STRING(80)" declaration).

Example of a string declaration with 35 characters:

```
str : STRING(35):= 'This is a String';
```

7.3.5 TIME

The data type is treated internally as DWORD. TIME is resolved in milliseconds.

Data type	Lower limit	Upper limit	Storage space	Resolution
TIME	T#0d0h0m0s0ms	T#49d17h2m47s295ms	32 bit	Milliseconds

7.3.6 LTIME

You can use the data type LTIME as a time base for high-resolution timer. A high-resolution timer has a resolution in nanoseconds.

Data Type	Lower Limit	Upper Limit	Memory
LTIME	LTIME#0NS	LTIME#213503D23H34M33S709MS551US615NS	64 bits

Syntax:

LTIME#<long time declaration>

The time declaration can include units of time that apply for the TIME constant as well as:

- "US": microseconds
- "NS": nanoseconds

Example:

LTIME1 := LTIME#1000D15H23M12S34MS2US44NS

7.3.7 Date and Time

The data types DATE, DATE_AND_TIME (DT), and TIME_OF_DAY (TOD) are handled internally like a DWORD (32-bit value).

The data types LDATE, LDATE_AND_TIME (LDT), and LTIME_OF_DAY (LTOD) are treated internally like an LWORD (64-bit value).

The values of these data types are measured in seconds, milliseconds, and nanoseconds since 01/01/1970.

Data Type	Lower Limit	Upper Limit	Memory	Resolution
DATE	DATE#1970-01-01	DATE#2106-02-07	32-bit	Seconds (although only the day is displayed)
	D#1970-01-01	D#2106-02-07		
DATE_AND_TIME	DATE_AND_TIME#1970-1-1-0:0:0	DATE_AND_TIME#2106-02-07-06:28:15	32-bit	Seconds
DT	DT#1970-1-1-0:0:0	DT#2106-02-07-06:28:15		
TIME_OF_DAY	TIME_OF_DAY#0:0:0	TIME_OF_DAY#23:59:59.999	32-bit	Milliseconds
TOD	TOD#0:0:0	TOD#23:59:59.999		
LDATE	LDATE#1970-1-1	LDATE#2554-7-21	64-bit	Nanoseconds (although only the day is displayed)
	LD#1970-1-1	LD#2554-7-2		
LDATE_AND_TIME	LDATE_AND_TIME#1970-1-1-0:0:0	LDATE_AND_TIME#2554-7-21:23:59:59.999999999	64-bit	Nanoseconds
LDT	LDT#1970-1-1-0:0:0	LDT#2554-7-21-23:59:59.999999999		
LTIME_OF_DAY	LTIME_OF_DAY#0:0:0	LTIME_OF_DAY#23:59:59.999999999	64-bit	Nanoseconds
LTOD	LTOD#0:0:0	LTOD#23:59:59.999999999		

7.3.8 BIT

The data type BIT is valid only in structures for the declaration of structure members or in a function block for the declaration of variables. A BIT variable can have the values TRUE (1) and FALSE (0). In this case, the variable requires exactly one bit of memory.

As a result, you can symbolically address individual bits by a name. BIT variables that are declared in succession are bundled in bytes. In this way, you can optimize memory use as opposed to BOOL types, which reserve 8 bits each. On the other hand, bit access is significantly more time-consuming. Therefore, you should use the BIT data type only when you need to define data in a predefined format.

7.3.9 Pointers

A pointer stores the memory address of objects, such as variables or function block instances, at runtime.

Syntax of the pointer declaration:

<pointer name>: POINTER TO <data type | data unit type | function block>;

Example

```
FUNCTION_BLOCK FB_Point
```

```
VAR
```

```
    piNumber: POINTER TO INT;
```

```
    iNumber1: INT := 5;
```

```
    iNumber2: INT;
```

```
END_VAR
```

```
piNumber := ADR(iNumber1); // piNumber is assigned to address of iNumber1
```

```
iNumber2 := piNumber^; // value 5 of iNumber1 is assigned to variable iNumber2 by dereferencing of pointer piNumber
```

Dereferencing a pointer means obtaining the value to which the pointer points. A pointer is dereferenced by appending the content operator `^` to the pointer identifier (for example, `piNumber^` in the example above). To assign the address of an object to a pointer, the address operator `ADR` is applied to the object: `ADR(iNumber1)`.

In online mode, you can click **Edit** → **Browse** → **Go to Reference** to jump from a pointer to the declaration location of the referenced variable.

Index access to pointers

CODESYS permits the index access `[]` to variables of type `POINTER TO`, as well as to the data types `STRING` or `WSTRING`.

The data, which the pointer points to, can also be accessed by appending the bracket operator `[]` to the pointer identifier (for example, `piData[i]`). The base data type of the pointer determines the data type and the size of the indexed component. In this case, the index access to the pointer is done arithmetically by adding the index dependent offset `i * sizeof(<base type>)` to the address of the pointer. The pointer is dereferenced implicitly at the same time.

Calculation: `piData[i] := (piData + i * sizeof(INT))^;`

This is **not**: `piData[i] := (piData + i)^.`

Index access STRING

When you use the index access with a variable of the type `STRING`, you get the character at the offset of the index expression. The result is of type `BYTE`. For example, `sData[i]` returns the *i*-th character of the character string `sData` as `SINT` (ASCII).

Index access WSTRING

When you use the index access with a variable of the type `WSTRING`, you get the character at the offset of the index expression. The result is of type `WORD`. For example, `wsData[i]` returns the “*i*”th character of the character string as `INT` (Unicode).

Subtracting pointers

The result of the difference between two pointers is a value of type `DWORD`, even on 64-bit platforms when the pointers are 64-bit pointers

7.3.10 ARRAY

An array is a collection of data elements of the same data type. CODESYS supports one- and multi-dimensional arrays of fixed or variable length.

Array of fixed length

You can define arrays in the declaration part of a POU or in global variable lists.

Syntax of the declaration of a one-dimensional array

<variable name> : ARRAY[<dimension>] OF <data type> (:= <initialization>)? ;
 <dimension> : <lower index bound>..<upper index bound>
 <data type> : elementary data types | user defined data types | function block types
 // (...) ? : Optional .

Syntax of the declaration of a multi-dimensional array

<variable name> : ARRAY[<1st dimension> (, <next dimension>)+] OF <data type> (:= <initialization>)? ;
 <1st dimension> : <1st lower index bound>..<1st upper index bound>
 <next dimension> : <next lower index bound>..<next upper index bound>
 <data type> : elementary data types | user defined data types | function block types
 // (...) + : One or more further dimensions
 // (...) ? : Optional
 The index limits are integers; maximum of the data type DINT.

Syntax for data access

<variable name>[<index of 1st dimension> (, <index of next dimension>)*]
 // (...) * : 0, one or more further dimensions

Example 1**One-dimensional array of 10 integer elements**

```
VAR
    aiCounter : ARRAY[0..9] OF INT;
END_VAR
```

Lower index limit: 0
 Upper index limit: 9

Initialization

```
aiCounter : ARRAY[0..9] OF INT := [0, 10, 20, 30, 40, 50, 60, 70, 80, 90];
```

Data access

```
iLocalVariable := aiCounter[2];
```

The value 20 is assigned to the local variable.

Example 2**2-dimensional array**

```
VAR
    aiCardGame : ARRAY[1..2, 3..4] OF INT;
END_VAR
```

1st dimension: 1 to 2
 2nd dimension: 3 to 4

Initialization

```
aiCardGame : ARRAY[1..2, 3..4] OF INT := [2(10),2(20)]; // Short notation for [10, 10, 20, 20]
```

Data access

```
iLocal_1 := aiCardGame[1, 3]; // Assignment of 10
```

```
iLocal_2 := aiCardGame[2, 4]; // Assignment of 20
```

Example 3**3-dimensional array**

```
VAR
```

```
    aiCardGame : ARRAY[1..2, 3..4, 5..6] OF INT;
```

```
END_VAR
```

1st dimension: 1 to 2

2nd dimension: 3 to 4

3rd dimension: 5 to 6

$2 * 2 * 2 = 8$ array elements

Initialization

```
aiCardGame : ARRAY[1..2, 3..4, 5..6] OF INT := [10, 20, 30, 40, 50, 60, 70, 80];
```

Data access

```
iLocal_1 := aiCardGame[1, 3, 5]; // Assignment of 10
```

```
iLocal_2 := aiCardGame[2, 3, 5]; // Assignment of 20
```

```
iLocal_3 := aiCardGame[1, 4, 5]; // Assignment of 30
```

```
iLocal_4 := aiCardGame[2, 4, 5]; // Assignment of 40
```

```
iLocal_5 := aiCardGame[1, 3, 6]; // Assignment of 50
```

```
iLocal_6 := aiCardGame[2, 3, 6]; // Assignment of 60
```

```
iLocal_7 := aiCardGame[1, 4, 6]; // Assignment of 70
```

```
iLocal_8 := aiCardGame[2, 4, 6]; // Assignment of 80
```

Initialization

```
aiCardGame : ARRAY[1..2, 3..4, 5..6] OF INT := [2(10), 2(20), 2(30), 2(40)]; // Short notation for [10, 10, 20, 20, 30, 30, 40, 40]
```

Data access

```
iLocal_1 := aiCardGame[1, 3, 5]; // Assignment of 10
```

```
iLocal_2 := aiCardGame[2, 3, 5]; // Assignment of 10
```

```
iLocal_3 := aiCardGame[1, 4, 5]; // Assignment of 20
```

```
iLocal_4 := aiCardGame[2, 4, 5]; // Assignment of 20
```

```
iLocal_5 := aiCardGame[1, 3, 6]; // Assignment of 30
```

```
iLocal_6 := aiCardGame[2, 3, 6]; // Assignment of 30
```

```
iLocal_7 := aiCardGame[1, 4, 6]; // Assignment of 40
```

```
iLocal_8 := aiCardGame[2, 4, 6]; // Assignment of 40
```

Example 4**3-dimensional arrays of a user-defined structure**

```

TYPE DATA_A
STRUCT
  iA_1 : INT;
  iA_2 : INT;
  dwA_3 : DWORD;
END_STRUCT
END_TYPE

PROGRAM PLC_PRG
VAR
  aData_A : ARRAY[1..3, 1..3, 1..10] OF DATA_A;
END_VAR

```

The array aData_A consists of a total of $3 * 3 * 10 = 90$ array elements of data type DATA_A.

Initialize partially

```

aData_A : ARRAY[1..3, 1..3, 1..10] OF DATA_A := [(iA_1 := 1, iA_2 := 10, dwA_3 := 16#00FF), (iA_1 := 2, iA_2 := 20, dwA_3 := 16#FF00), (iA_1 := 3, iA_2 := 30, dwA_3 := 16#FFFF)];

```

In the example, only the first 3 elements are initialized explicitly. Elements to which no initialization value is assigned explicitly are initialized internally with the default value of the basic data type. This initializes the structure components at 0 starting with the element aData_A[2, 1, 1].

Data access

```

iLocal_1 := aData_A[1,1,1].iA_1; // Assignment of 1
dwLocal_2 := aData_A[3,1,1].dwA_3; // Assignment of 16#FFFF

```

Example 5**Array of a function block**

```

FUNCTION BLOCK FBlockObject_A
VAR
  iCounter : INT;
END_VAR
...

PROGRAM PLC_PRG
VAR
  aObject_A : ARRAY[1..4] OF FBlockObject_A;
END_VAR

```

The array aObject_A consists of 4 elements. Each element instantiates a FBlockObject_A function block.

Function call

```
aObject_A[2]();
```

Example 6

Implementation of FB_Something with method FB_Init

```
FUNCTION_BLOCK FB_Something
```

```
VAR
```

```
  _nId : INT;
```

```
  _lrln : LREAL;
```

```
END_VAR
```

```
...
```

```
METHOD FB_Init : BOOL
```

```
VAR_INPUT
```

```
  bInitRetains : BOOL;
```

```
  bInCopyCode : BOOL;
```

```
  nId : INT;
```

```
  lrln : LREAL;
```

```
END_VAR
```

```
_nId := nId;
```

```
_lrln := lrln;
```

The function block FB_Something has a method FB_Init that requires 2 parameters.

Instantiation of the array with initialization

```
PROGRAM PLC_PRG
```

```
VAR
```

```
  fb_Something_1 : FB_Something(nId := 11, lrln := 33.44);
```

```
  a_Something : ARRAY[0..1, 0..1] OF FB_Something[(nId := 12, lrln := 11.22), (nId := 13, lrln := 22.33), (nId := 14, lrln := 33.55), (nId := 15, lrln := 11.22)];
```

```
END_VAR
```

7.3.10.1 Array of arrays

The declaration of an "array of arrays" is an alternative syntax for multidimensional arrays. A collection of elements is nested instead of dimensioning the elements. The nesting depth is unlimited.

Syntax for declaration

```
<variable name> : ARRAY[<first>] ( OF ARRAY[<next>] )+ OF <data type> ( := <initialization> )? ;
```

```
<first> : <first lower index bound>..<first upper index bound>
```

```
<next> : <lower index bound>..<upper index bound> // one or more arrays
```

```
<data type> : elementary data types | user defined data types | function block types
```

```
// (...) + : One or more further arrays
```

```
// (...) ? : Optional
```

Syntax for data access

```
<variable name>[<index of first array>] ( [<index of next array>] )+ ;
```

```
// (...) * : 0, one or more further arrays
```

Example 7

```
PROGRAM PLC_PRG
```

```
VAR
```

```
    aiPoints : ARRAY[1..2,1..3] OF INT := [1,2,3,4,5,6];
```

```
    ai2Boxes : ARRAY[1..2] OF ARRAY[1..3] OF INT := [ [ 1, 2, 3], [ 4, 5, 6] ];
```

```
    ai3Boxes : ARRAY[1..2] OF ARRAY[1..3] OF ARRAY[1..4] OF INT := [ [ [ 1, 2, 3, 4], [5, 6, 7, 8 ], [9, 10, 11, 12] ], [ [13, 14, 15, 16], [ 17, 18, 19, 20], [21, 22, 23, 24] ] ];
```





















```
    ai4Boxes : ARRAY[1..2] OF ARRAY[1..3] OF ARRAY[1..4] OF ARRAY[1..5] OF INT;
```

```
END_VAR
```

```
aiPoints[1, 2] := 1200;
```

```
ai2Boxes[1][2] := 1200;
```

The variables aiPoints and ai2Boxes collect the same data elements, however the syntax for the declaration differs from that of the data access.

		aiPoints	ARRAY [1..2, 1..3] OF INT	
		aiPoints[1, 1]	INT	1
		aiPoints[1, 2]	INT	1200
		aiPoints[1, 3]	INT	3
		aiPoints[2, 1]	INT	4
		aiPoints[2, 2]	INT	5
		aiPoints[2, 3]	INT	6
		ai2Boxes	ARRAY [1..2] OF ARRAY [1..3] OF INT	
			ai2Boxes[1]	ARRAY [1..3] OF INT
			ai2Boxes[1][1]	INT
			ai2Boxes[1][2]	INT
			ai2Boxes[1][3]	INT
			ai2Boxes[2]	ARRAY [1..3] OF INT
			ai2Boxes[2][1]	INT
			ai2Boxes[2][2]	INT
			ai2Boxes[2][3]	INT

Array of variable length

In function blocks, functions, or methods, you can declare arrays of variable length in the VAR_IN_OUT declaration section.

The LOWER_BOUND and UPPER_BOUND operators are provided for determining the index limits of the actual used array at runtime.

Syntax of the declaration of a one-dimensional array of variable length

```
<variable name> : ARRAY[*] OF <data type> ( := <initialization> )? ;
```

```
<data type> : elementary data types | user defined data types | function block types
```

```
// (...)?: Optional
```

Syntax of the declaration of a multi-dimensional array of variable length

```
<variable name> : ARRAY[* ( , * )+ ] OF <data type> ( := <initialization> )? ;
```

```
<data type> : elementary data types | user defined data types | function block types
```

```
// (...)+: One or more further dimensions
```

// (...)?: Optional

Syntax of the operators for calculating the limit index

LOWER_BOUND(<variable name> , <dimension number>)

UPPER_BOUND(<variable name> , <dimension number>)

Example 8

The SUM function adds the integer values of the array elements and returns the calculated sum as a result. The sum is calculated across all array elements available at runtime. As the actual number of array elements will only be known at runtime, the local variable is declared as a one-dimensional array of variable length.

```
FUNCTION SUM: INT;
VAR_IN_OUT
    aiData : ARRAY[*] OF INT;
END_VAR
VAR
    diCounter, diResult : DINT;
END_VAR
diResult := 0;
FOR diCounter := LOWER_BOUND(aiData, 1) TO UPPER_BOUND(aiData, 1) DO // Calculates
the length of the current array
    diResult := diResult + A[i];
END_FOR;
```

7.3.11 SUM := diResult;Structure (STRUCT)

A structure is a user-defined data type, which combines multiple variables of any data type into a logical unit. The variables declared within a structure are called members.

You make the type declaration of a structure in a “DUT” object which you create in the Project → Add Object → DUT menu or in the context menu of an application.

Syntax

TYPE <structure name> :

STRUCT

(<variable declaration optional with initialization>)+

END_STRUCT

END_TYPE

<structure name> is an identifier which is valid in the entire project so that you can use it like a standard data type. Moreover, you can declare any number of variables (at least one) which are supplemented optionally by an initialization.

Structures can also be nested. This means that you declare a structure member with an existing structure type. Then the only restriction is that you must not assign any address to the variable (structure member). (The AT declaration is not permitted here.)

Example 1

Type declaration

```
TYPE S_POLYGONLINE :
```

```

STRUCT
    aiStart : ARRAY[1..2] OF INT := [-99, -99];
    aiPoint1 : ARRAY[1..2] OF INT;
    aiPoint2 : ARRAY[1..2] OF INT;
    aiPoint3 : ARRAY[1..2] OF INT;
    aiPoint4 : ARRAY[1..2] OF INT;
    aiEnd : ARRAY[1..2] OF INT := [99, 99];
END_STRUCT
END_TYPE

```

Extension of a type declaration

An additional structure is declared from an existing structure. In addition to its own members, the extended structure also has the same structure members as the base structure.

Syntax

```

TYPE <structure name> EXTENDS <basis structure> :
STRUCT
    ( <variable declaration optional with initialization> )+
END_STRUCT
END_TYPE

```

Example 2

Type declaration

```

TYPE S_PENTAGON EXTENDS S_POLYGONLINE :
STRUCT
    aiPoint5 : ARRAY[1..2] OF INT;
END_STRUCT
END_TYPE

```

Declaration and initialization of structure variables

Example

```

PROGRAM progLine
VAR
    sPolygon : S_POLYGONLINE := (aiStart:=[1,1], aiPoint1:=[5,2], aiPoint2:=[7,3], aiPoint3:=[8,5],
    aiPoint4:=[5,7], aiEnd:=[1,1]);
    sPentagon : S_PENTAGON := (aiStart:=[0,0], aiPoint1:=[1,1], aiPoint2:=[2,2], aiPoint3:=[3,3],
    aiPoint4:=[4,4], aiPoint5:=[5,5], aiEnd:=[0,0]);
END_VAR

```

You must not permitted to use initializations with variables. For an example of initializing an array of a structure, see the help page for the data type ARRAY.

Access to a structure member

You access structure members with the following syntax:

```
<variable name> . <component name>
```

Example 3

```

PROGRAM prog_Polygon
VAR
    sPolygon : S_POLYGONLINE := (aiStart:=[1,1], aiPoint1:=[5,2], aiPoint2:=[7,3], aiPoint3:=[8,5],
    aiPoint4:=[5,7], aiEnd:=[1,1]);
    iPoint: INT;
END_VAR
// Assigns 5 to aiPoint
iPoint := sPolygon.aiPoint1[1];
Result: iPoint = 5

```

Symbolic bit access in structure variables

You can declare a structure with variables of data type BIT to combine individual bits into a logical unit. Then you can symbolically address individual bits by a name (instead of by a bit index).

Syntax declaration

```

TYPE <structure name> :
STRUCT
    ( <bit name> : BIT; )+
END_STRUCT
END_TYPE

Syntax of bit access

<structure name> . <bit name>

```

Example 4**Type declaration**

```

TYPE S_CONTROL :
STRUCT
    bitOperationEnabled : BIT;
    bitSwitchOnActive : BIT;
    bitEnableOperation : BIT;
    bitError : BIT;
    bitVoltageEnabled : BIT;
    bitQuickStop : BIT;
    bitSwitchOnLocked : BIT;
    bitWarning : BIT;
END_STRUCT
END_TYPE

```

Bit access

```

FUNCTION_BLOCK FB_Controller
VAR_INPUT
    xStart : BOOL;

```



```

END_VAR
VAR_OUTPUT
END_VAR
VAR
    ControlDriveA : S_CONTROL;
END_VAR
IF xStart = TRUE THEN
    // Symbolic bit access
    ControlDriveA.bitEnableOperation := TRUE;
END_IF
PROGRAM PLC_PRG
VAR
    fbController : FB_Controller;
END_VAR
fbController();
fbController.xStart := TRUE;

```

7.3.12 Enumerations (ENUM)

An enumeration is a user-defined data type composed of a series of comma-separated components (enumeration values) for declaring user-defined variables. Moreover, you can use the enumeration components like constants whose identifier <enumeration name>.<component name> is recognized globally in the project.

You declare an enumeration in a DUT object, which you have already created in the project by clicking “Add Object”.

Declaration

Syntax

```

( {attribute 'strict'} )? // Pragma optional but recommended
TYPE <enumeration name> :
(
    <first component declaration>,
    ( <component declaration> ,)+
    <last component declaration>
)( <basic data type> )? ( := <default variable initialization> )? ;
END_TYPE

```

(...)? : Optional

<component declaration> : <component name> (:= <component initialization>)?

<basic data type> : INT | UINT | SINT | USINT | DINT | UDINT | LINT | ULINT | BYTE | WORD | DWORD | LWORD

<variable initialization> : <one of the component names>

In an enumeration declaration, at least 2 components are usually declared. However, you can declare as many as you want. Every single component can be assigned its own initialization. Enumerations automatically have the basic data type INT, but you can specify another basic data type. Moreover, you can specify a component in the declaration with which an enumeration variable is then initialized.

The pragma {attribute 'strict'} causes a strict type test to be performed as described below.

Example 1

```
{attribute 'qualified_only'}
{attribute 'strict'}
TYPE COLOR_BASIC :
(
    yellow,
    green,
    blue,
    black
)
; // Basic data type is INT, default initialization for all COLOR_BASIC variables is yellow
END_TYPE
```

Enumeration with explicit basic data type

Extensions to the IEC 61131-3 standard

The basic data type for an enumeration declaration is INT by default. However, you can also declare enumerations that are based explicitly on another integer data type.

<basic data type> : INT | UINT | SINT | USINT | DINT | UDINT | LINT | ULINT | BYTE | WORD | DWORD | LWORD

Example 2

Enumeration with basic data type DWORD

```
TYPE COLOR :
(
    white := 16#FFFFFF00,
    yellow := 16#FFFFFF00,
    green := 16#FF00FF00,
    blue := 16#FF0000FF,
    black := 16#88000000
) DWORD := black; // Basic data type is DWORD, default initialization for all COLOR variables
is black
END_TYPE
```

The strict programming rules are activated when adding the pragma {attribute 'strict'}.

.The following code is considered a compiler error:

Arithmetic operations with enumeration components

For example, an enumeration variable cannot be used as a counter variable in a FOR loop.

Assignment of a constant value, which does not correspond to an enumeration value, to an enumeration component

Assignment of a non-constant variable, which has another data type as the enumeration, to an enumeration component

Arithmetic operations can lead to undeclared values being assigned to enumeration components. A better programming style is to use SWITCH/CASE statements for processing component values.

Declaration and initialization of enumeration variables

Syntax

```
<variable name> : <enumeration name> ( := <initialization> )? ;
```

For a declaration of an enumeration variable with user-defined data type <enumeration name>, this can be initialized with an enumeration component.

Example 3

```
PROGRAM PLC_PRG
VAR
    colorCar: COLOR;
    colorTaxi : COLOR := COLOR.yellow;
END_VAR
```

The variable colorCar is initialized with COLOR.black. That is the default initialization for all enumeration variables of type COLOR and defined this way in the type declaration. The variable colorTaxi has its own initialization.

If no initializations are specified, then the initialization value is 0.

Example 4

```
PROGRAM PLC_PRG
VAR
    cbFlower : COLOR_BASIC;
    cbTree: COLOR_BASIC := COLOR_BASIC.green;
END_VAR
```

The variable cbFlower is initialized with COLOR_BASIC.yellow. That is the default initialization for all enumeration variables of type COLOR_BASIC. Because the enumeration declaration does not specify a component for initialization, the system automatically initializes with the component that has the value 0. This is usually the first of the enumeration components. However, it can also be another component that is not in the first position but explicitly initialized with 0.

The variable cbTree has an explicit initialization.

If no value is specified for both the type and the variable, then the following rule applies: If an enumeration contains a value for 0, then this value is the default initialization, and if not, then the first component in the list.

Example 5

Initialization with the 0 component

```
TYPE ENUM :
```

```
(
    e1 := 2,
    e2 := 0,
    e3
)
;
END_TYPE
```

```
PROGRAM PLC_PRG
```

```
VAR
```

```
    e : ENUM;
```

```
END_VAR
```

The variable e is initialized with ENUM.e2.

Initialization with the first component

```
TYPE ENUM2 :
```

```
(
    e1 := 3,
    e2 := 1,
    e3
)
;
END_TYPE
```

```
PROGRAM PLC_PRG
```

```
VAR
```

```
    e2 : ENUM2;
```

```
END_VAR
```

The variable e2 is initialized with ENUM.e1.

Unique access to enumeration components

Extensions to the IEC 61131-3 standard

The enumeration components can also be used as constant variables with the identifier <enumeration name>.<component name>. Enumeration components are recognized globally in the project and access to them is unique. Therefore, a component name can be used in different enumerations.

Example 6

Component blue

```
PROGRAM PLC_PRG
```

```
VAR
```

```
    cbFlower : COLOR_BASIC;
```

```
    colorCar : COLOR;
```

```
END_VAR
```

(* unambiguous identifiers although the component names are identical *)

cbFlower := COLOR_BASIC.blue;

colorCar := COLOR.blue;

(* invalid code *)

cbFlower := blue;

colorCar := blue;

7.4 ST Statements

7.4.1 IF

The IF statement is used for checking a condition and, depending on this condition, for executing the subsequent statements.

A condition is coded as an expression that returns a Boolean value. If the expression returns TRUE, then the condition is fulfilled and the corresponding statements after THEN are executed. If the expression returns FALSE, then the following conditions, which are identified with ELSIF, are evaluated. If an ELSIF condition returns TRUE, then the statements are executed after the corresponding THEN. If all conditions return FALSE, then the statements after ELSE are executed.

Therefore, at most one branch of the IF statement is executed. ELSIF branches and the ELSE branch are optional.

SyntaxIF <condition> THEN

<statements>

(ELSIF <condition> THEN

<statements>)*

(ELSE

<statements>)?

END_IF;

// (...) * None, once or several times

// (...) ? Optional

Example

PROGRAM PLC_PRG

VAR

iTemp: INT;

xHeatingOn: BOOL;

xOpenWindow: BOOL;

END_VAR

IF iTemp < 17 THEN

xHeatingOn := TRUE;

ELSIF iTemp > 25 THEN

xOpenWindow := TRUE;

ELSE xHeatingOn := FALSE;

END_IF;

The program is run as follows at runtime:

For the evaluation of the expression `iTemp < 17 = TRUE`, the subsequent statement is executed and the heating is switched on. For the evaluation of the expression `iTemp < 17 = FALSE`, the subsequent ELSIF condition `iTemp > 25` is evaluated. If this is true, then the statements in ELSIF are executed and the view is opened. If all conditions are FALSE, then the statement in ELSE is executed and the heating is switched off.

7.4.2 FOR

The FOR loop is used to execute instructions with a certain number of repetitions.

Syntax:

```
FOR <counter> := <start value> TO <end value> {BY <increment> } DO
  <instructions>
END_FOR;
```

The section inside the curly parentheses {} is optional.

CODESYS executes the <instructions> as long as the <counter> is not greater, or - in case of negative increment - is not smaller than the <end value>. This is checked before the execution of the <instructions>.

Every time the instructions <instructions> have been executed, the counter <counter> is automatically increased by the increment <increment>. The increment <increment> can have any integral value. If you do not specify an increment, the standard increment is 1.

Example

```
FOR iCounter := 1 TO 5 BY 1 DO
  iVar1 := iVar1*2;
END_FOR;
Erg := iVar1;
```

If you have pre-configured iVar1 with 1, iVar1 has the value 32 after the FOR loop.

The end value <end value> may not attain the same value as the upper limit of the data type of the counter.

If the end value of the counter is equal to the upper limit of the data type of the counter, an endless loop results. For example, an endless loop results in the above example if iCounter is of the data type SINT and the <end value> equals 127, since the data type SINT has the upper limit 127.

7.4.3 CASE

Use this dialog box for pooling several conditional instructions containing the same condition variable into a construct.

Syntax:

```
CASE <Var1> OF
  <value1>:<instruction1>
  <value2>:<instruction2>
  <value3, value4, value5>:<instruction3>
  <value6 ... value10>:<instruction4>
  ...
  <value n>:<instruction n>
```

```
{ELSE <ELSE-instruction>}
```

```
END_CASE;
```

The section within the curly brackets {} is optional.

Processing scheme of a CASE instruction.

If the value of the variable <Var1> is <value i>, then the instruction <instruction i> is executed.

If the variable <Var1> has non of the given values, then the <ELSE-instruction> is executed.

If the same instruction is executed for several values of the variable, then you can write the values in sequence, seperated by commas.

Example 1

```
CASE iVar OF
  1, 5: bVar1 := TRUE;
      bVar3 := FALSE;

  2: bVar2 := FALSE;
      bVar3 := TRUE;

  10..20: bVar1 := TRUE;
          bVar3= TRUE;
ELSE
  bVar1 := NOT bVar1;
  bVar2 := bVar1 OR bVar2;
END_CASE;
```

7.4.4 WHILE

The WHILE loop is used like the FOR loop in order to execute instructions several times until the abort condition occurs. The abort condition of a WHILE loop is a boolean expression.

Syntax:

```
WHILE <boolean expression> DO
  <instructions>
END_WHILE;
```

CODESYS repeatedly executes the <instructions> for as long as the <boolean expression> returns TRUE. If the boolean expression is already FALSE at the first evaluation, then CODESYS never executes the instructions. If the boolean expression never adopts the value FALSE, then the instructions are repeated endlessly, as a result of which a runtime error results.

Example

```
WHILE iCounter <> 0 DO
  Var1 := Var1*2
  iCounter := iCounter-1;
END_WHILE;
```

You must ensure by programming means that no endless loops are caused.

In a certain sense the WHILE and REPEAT loops are more powerful than the FOR loop, since you don't need to already know the number of executions of the loop before its execution. In some cases it is thus only possible to work with these two kinds of loop. If the number of executions of the loop is clear, however, then a FOR loop is preferable in order to avoid endless loops.

7.5 REPEAT

The REPEAT loop is used like the WHILE loop, but with the difference that CODESYS only checks the abort condition after the execution of the loop. The consequence of this behavior is that the REPEAT loop is executed at least once, regardless of the abort condition.

Syntax:

```
REPEAT
<instructions>
UNTIL <boolean expression>
END_REPEAT;
```

CODESYS executes the <instructions> until the <boolean expression> returns TRUE.

If the boolean expression already returns TRUE at the first evaluation, CODESYS executes the instructions precisely once. If the boolean expression never adopts the value TRUE, then the instructions are repeated endlessly, as a result of which a runtime error results.

Example

```
REPEAT
Var1 := Var1*2;
iCounter := iCounter-1;
UNTIL
iCounter = 0
END_REPEAT;
```

In a certain sense the WHILE and REPEAT loops are more powerful than the FOR loop, since the number of executions of the loop doesn't already need to be known before its execution. In some cases you can only work with these two kinds of loop. If the number of executions of the loop is clear, however, then a FOR loop is preferable in order to avoid endless loops.

7.5.1 RETURN

Use the RETURN statement in order to exit from a function block. You can make this dependent on a condition, for example.

Example

```
IF xIsDone = TRUE THEN
RETURN;
END_IF;
iCounter := iCounter + 1;
```

If the value of xIsDone is equal to TRUE, then the function block is exited immediately and the statement iCounter := iCounter + 1; is not executed.

7.5.2 JMP

The JMP instruction is used to execute an unconditional jump to a program line that is marked by a jump label.

Syntax:


```
<label>: <instructions>
```

```
JMP <label>;
```

The jump label <label> is any unique identifier that you place at the beginning of a program line. On reaching the JMP instruction, a return to the program line with the <label> takes place.

Example

```
iVar1 := 0;
_label1: iVar1 := iVar1+1;
(*instructions*)

IF (iVar1 < 10) THEN
JMP _label1;
END_IF;
```

7.5.3 EXIT

The EXIT instruction is used in a FOR, WHILE or REPEAT loop in order to end the loop regardless of other abort conditions.

7.5.4 CONTINUE

CONTINUE is an instruction of the Extended Structured Text (ExST).

The instruction is used inside FOR, WHILE and REPEAT loops in order to jump to the beginning of the next execution of the loop.

Example

```
FOR Counter:=1 TO 5 BY 1 DO
INT1:=INT1/2;
IF INT1=0 THEN
CONTINUE; (* to avoid a division by zero *)
END_IF
Var1:=Var1/INT1; (* executed, if INT1 is not 0 *)
END_FOR;
Erg:=Var1;
```

7.5.5 Function Block Call

Syntax

```
<FB-instance>(<FB input variable>:=<value or address>|, <other FB input variables>);
```

Example

```
TMR:TON;
TMR (IN:=%OX5, PT:=T#300ms);
varA:=TMR.Q;
```

The timer function block TON is instanced in TMR:TON and called with assignments for the parameters IN and PT.

The output Q is addressed with TMR.Q and assigned to the variable varA.

8 MOTION SOLUTION PROJECT

This section is an introduction to the key features and tools, which will be used during the Motion Solution Project.

8.1 Introduction

8.1.1 Understanding the Motion Solution Project

The project type “Motion Solution Project” is an AC500 project with inbuilt wizards which help the user by intuitively guiding step by step process to add the motion axis and help the user to configure it in few simple steps. These include code writing, adding Hardware and intuitively setting up PDO mapping.

8.1.2 Understanding the Motion Solution Wizard

The Motion Solution wizard is a tool which helps the user to efficiently configure the motion axis. The motion wizard helps the user to configure the axis based on PTO or EtherCAT using Automation Builder . Afterwards users can proceed by adding further PLCopen function blocks based on their application needs.

8.1.3 Understanding the Axis Objects

When using the motion solution to commission the user will come across the axis objects. These are program elements which allow the user to;

- define the motor drive type,
- mechanical units, gearing, and Scaling
- control settings
- PLC Limits (Speed, Acceleration etc)
- PDO mapped data (Only for EtherCAT axis)
- Drive Control Mode (Only for EtherCAT axis)

8.2 Installing the latest Motion Control Wizard and Libraries

As described earlier, during the Automation Builder installation process the user will be prompted to select and install “Motion Control (PS5611)”. As long as this step is carried out the user will have the latest version of libraries and Motion control tools such as the Motion Control wizard and Cam editor.

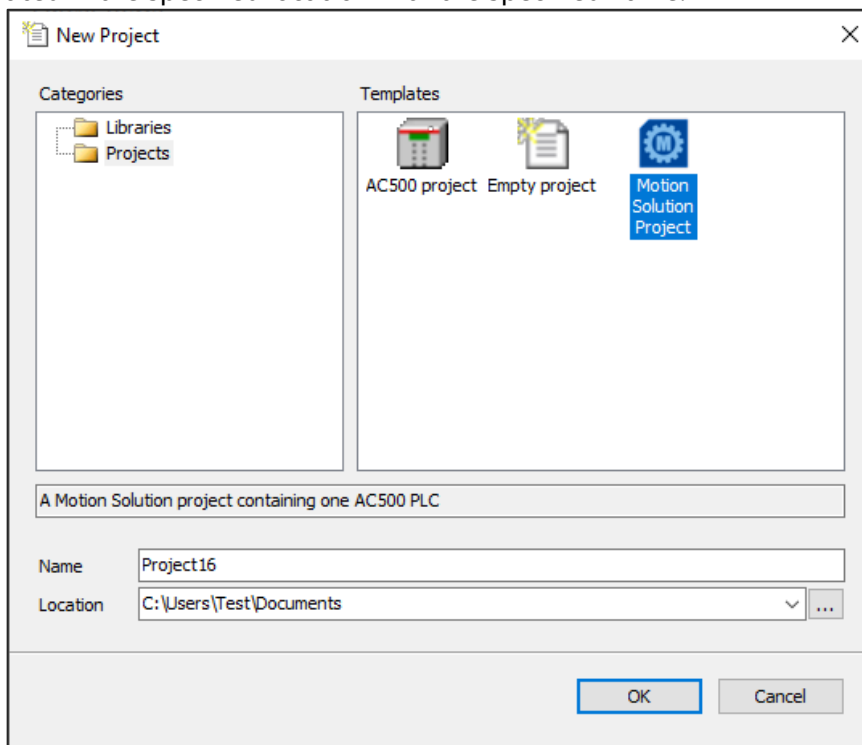
8.3 Creating new Motion Solution project

Now we understand the installation process and the elements that will be used in the configuration we can go through the process of using the “Motion Solution Project” to create a new configuration.

8.3.1 Creating new project

Start Automation Builder and select “New Project”.

Select the “Motion Solution Project” icon as shown below. Click “OK” button and a new project will be created in the specified location with the specified name.



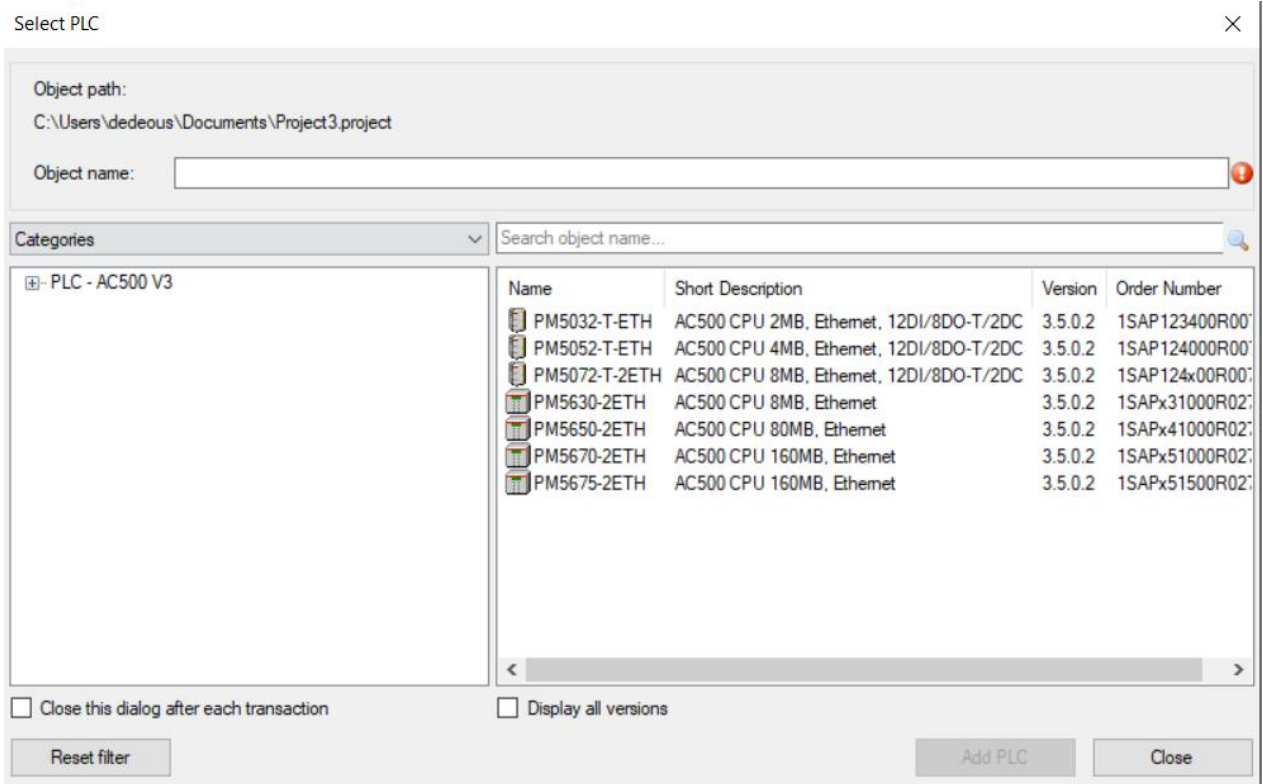
8.3.2 Add PLC types

Automation Builder will now pop up a “Select PLC” window and from here user can select one of the V3 PLC.

Select one of the AC500 PLC type and click “Add PLC” button to get it added to Automation Builder hardware tree.

The Motion Solution Wizard then assumes automatically the 1 slot Terminal base and Ethercat coupler in addition the PLC chosen, (these)are included in the PS56xx-MC-Kits.

Note: The PLC type can be changed at any time using the steps outlined earlier.



PM50xx-T-ETH support only PTO axis and PM56xx-2ETH support only EtherCAT axis.

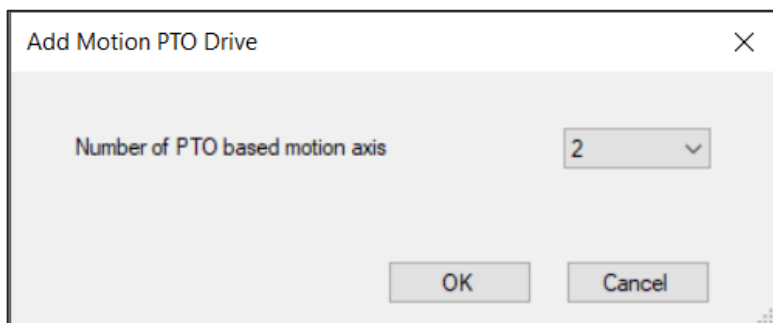


Note: PM5012-x-ETH and all the PM50xx-R-ETH eCo PLC are not supported by motion solution wizard.

After selecting the PLC type, Motion Solution Wizard will guide the user to add PTO / EtherCAT axis based on the selected PLC type.

8.3.3 Add PTO axis

After creating the hardware tree, Automation Builder will now pop up “Add Motion PTO Drive” window. Here user can configure the number of PTO drives needed for the application. A maximum of four PTO axis can be configured based on the PTO axis frequency.



User can also add the PTO axis by right clicking on the “OnBoard_IO” object and select the “Add Object” option or by using the Add PTO axis button from “Motion Solution Wizard” overview page.

Due to differences in the performance of CPU types, there are different limits on:
the minimum cycle time configurable in each PLC type.

PLC Type	PM5032	PM5052	PM5072
Min. cycle time	5 ms	2 ms	1 ms

Number of PTO axis per PLC

Below are the maximum PTO axes combinations possible based on the PTO frequency configured.

- Max 4 PTO axis - 100Khz
- Max 3 PTO axis - Two 100Khz & one 200Khz
- Max 2 PTO axis - 200Khz

100Khz (default) PTO axis

When the axis is configured as 100Khz, output channel O0 to O3 (Value = PTOx Dir) are configured as direction output and O4 to O7 (Value = PTOx LS Pulse) are configured as pulse. Users need to connect the PTI drive cables accordingly.

Below table shows the user an overview of the hardware channels configured based on the number of axes configured, this can be used as a reference for the pulse and direction wiring to PTI drive.

		Axis frequency in Khz				
		Axis1	100	100	100	100
		Axis2		100	100	100
		Axis3			100	100
		Axis4				100
HW Channel Selection	DO0	PWM/PTO0 Dir	PWM/PTO0 Dir	PWM/PTO0 Dir	PWM/PTO0 Dir	PWM/PTO0 Dir
	DO1	PWM/PTO1 Dir		PWM/PTO1 Dir	PWM/PTO1 Dir	PWM/PTO1 Dir
	DO2	PWM/PTO2 Dir			PWM/PTO2 Dir	PWM/PTO2 Dir
	DO3	PWM/PTO3 Dir				PWM/PTO3 Dir
	DO4	PWM/PTO0 LS Pulse PTO0 HS Pulse / Cw	PWM/PTO0 LS Pulse	PWM/PTO0 LS Pulse	PWM/PTO0 LS Pulse	PWM/PTO0 LS Pulse
	DO5	PWM/PTO1 LS Pulse PTO0 HS Dir / Ccw		PWM/PTO1 LS Pulse	PWM/PTO1 LS Pulse	PWM/PTO1 LS Pulse
	DO6	PWM/PTO2 LS Pulse PTO1 HS Pulse / Cw			PWM/PTO2 LS Pulse	PWM/PTO2 LS Pulse
	DO7	PWM/PTO3 LS Pulse PTO1 HS Dir / Ccw				PWM/PTO3 LS Pulse

For example, when two PTO axes are configured as 100Khz, the motion solution wizard set the onboard output configuration. For the first axis, direction as Output0 and pulse as Output4 and for the second axis direction as Output1 and pulse as Output5.

200Khz PTO axis

When the axis is configured as 200Khz, output channel O4 (Value = PTOx HS Pulse) and O5 (Value = PTOx HS Dir) are configured as pulse and direction output for the first axis and O6 and O7 is configured as pulse and direction output for the second axis.

Below table shows the user an overview of the hardware channels configured based on the number of axes configured, this can be used as a reference for the pulse and direction wiring to PTI drive.

		Axis frequency in Khz		
		Axis1	200	200
		Axis2		200
		Axis3		
		Axis4		
HW Channel Selection	DO0	PWM/PTO0 Dir		
	DO1	PWM/PTO1 Dir		
	DO2	PWM/PTO2 Dir		
	DO3	PWM/PTO3 Dir		
	DO4	PWM/PTO0 LS Pulse PTO0 HS Pulse / Cw	PTO0 HS Pulse / Cw	PTO0 HS Pulse / Cw
	DO5	PWM/PTO1 LS Pulse PTO0 HS Dir / Ccw	PTO0 HS Dir / Ccw	PTO0 HS Dir / Ccw
	DO6	PWM/PTO2 LS Pulse PTO1 HS Pulse / Cw		PTO1 HS Pulse / Cw
	DO7	PWM/PTO3 LS Pulse PTO1 HS Dir / Ccw		PTO1 HS Dir / Ccw

For example, when two PTO axes are configured as 200Khz, the motion solution wizard set the onboard output configuration. For the first axis, Pulse as Output4 and direction as Output5 and for the second axis pulse as Output6 and direction as Output7.

100Khz and 200Khz PTO axis

When some axes are configured as 100Khz and some are configured as 200Khz frequency, user must take care following points,

Make sure the 200Khz axis is configured as first or last axis and not in between 100Khz axis.

When the first axis is 200Khz, Automation Builder will configure the O4 and O5 channels and for 100Khz O6 and O7 is configured as Pulse and O2 & O3 are configured as direction.

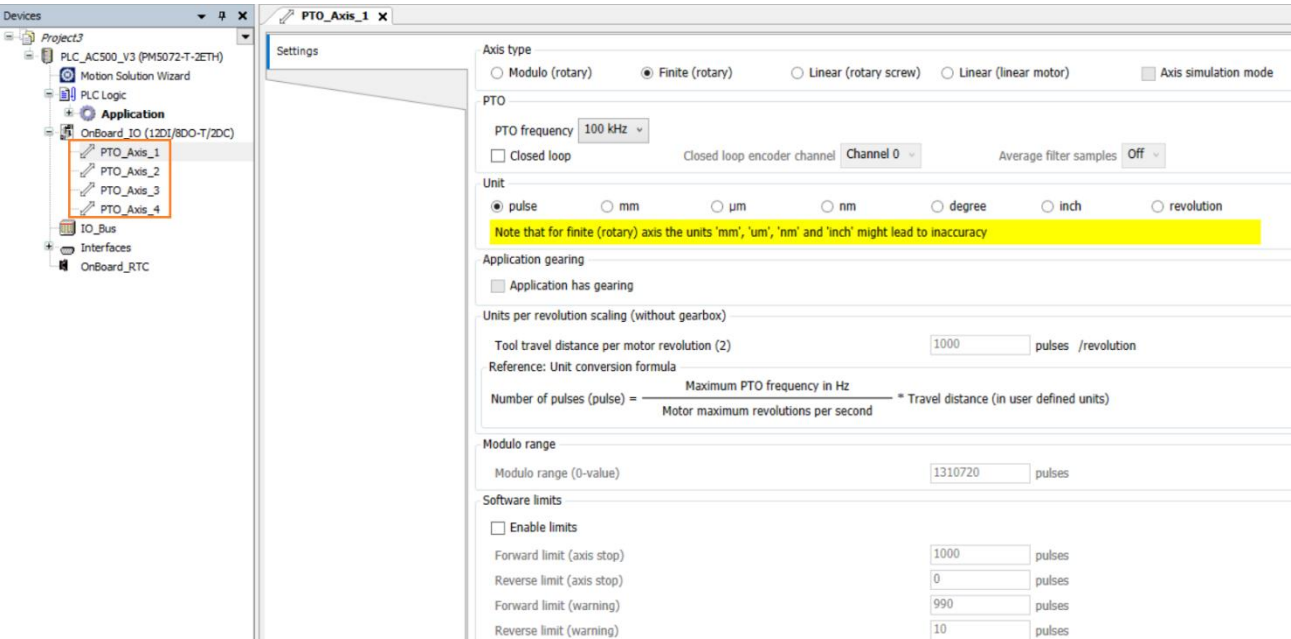
Below table shows the user an overview of the hardware channels configured based on the number of axes configured.

		Axis frequency in Khz					
		Axis1	100	200	100	200	100
		Axis2	200	100	100	100	200
		Axis3			200	100	100
		Axis4					
HW Channel Selection	DO0	PWM/PTO0 Dir	PWM/PTO0 Dir		PWM/PTO0 Dir		
	DO1	PWM/PTO1 Dir			PWM/PTO1 Dir		
	DO2	PWM/PTO2 Dir		PWM/PTO2 Dir		PWM/PTO2 Dir	
	DO3	PWM/PTO3 Dir				PWM/PTO3 Dir	
	DO4	PWM/PTO0 LS Pulse PTO0 HS Pulse / Cw	PWM/PTO0 LS Pulse	PTO0 HS Pulse / Cw	PWM/PTO0 LS Pulse	PTO0 HS Pulse / Cw	Configuration not allowed. Please keep 200Khz axis as first or last axis
	DO5	PWM/PTO1 LS Pulse PTO0 HS Dir / Ccw		PTO0 HS Dir / Ccw	PWM/PTO1 LS Pulse	PTO0 HS Dir / Ccw	
	DO6	PWM/PTO2 LS Pulse PTO1 HS Pulse / Cw	PTO1 HS Pulse / Cw	PWM/PTO2 LS Pulse	PTO1 HS Pulse / Cw	PWM/PTO2 LS Pulse	
	DO7	PWM/PTO3 LS Pulse PTO1 HS Dir / Ccw	PTO1 HS Dir / Ccw		PTO1 HS Dir / Ccw	PWM/PTO3 LS Pulse	

8.3.3.1 PTO Axis Object

Nested underneath the OnBoard_IO object is the PTO Axis object. From here users can configure each axis separately as per the application requirement by opening the axis object which is added under the OnBoard_IO. To do this double click on the object to open “Settings” page. Here user can update settings as per the application requirement.

Later when downloading the application, the wizard will use these settings to define the operation and scaling of this axis object.



8.3.3.2 PTO Axis Object settings

All settings related to the application and axis specific will be done here and needs to be carefully up- dated for each axis. Based on the inputs provided here, wizard will compile and generated the code.

Axis Type

The user can select the type of Axis to be configured based on the application requirement. Below is the list of settings which user can configure from the Motion Control wizard along with their meanings.

Axis type	
<input checked="" type="radio"/> Modulo (rotary) <input type="radio"/> Finite (rotary) <input type="radio"/> Linear (rotary screw) <input type="radio"/> Linear (linear motor) <input type="checkbox"/> Axis simulation mode	
Modulo (rotary)	Default setting in the wizard. By selecting the Modulo (Rotary) your axis will be configured as a roll-over axis and the desired modulo range can be configured later.
Finite (rotary)	Your axis will be configured as a roll-over axis where in modulo range is non editable by the user and calculated based on the “Unit” selection, Inc_Per_R, U_Per_Rev_Nominator and U_Per_Rev_Denominator setting.
Linear (rotary screw)	This needs to be configured when the user having a rotary motor with linear movements (linear axis).
Linear (linear motor)	This needs to be configured when the user axis is a linear motor.
Axis simulation mode	This option is read only from here and needed when the user config- ured the axis but not have the real hardware yet. This can be selected from the motion solution wizard overview page. For virtual axis configuration please refer to the chapter “Add and con- figure virtual axis”.

PTO

User can set the PTO axis related settings here: i.e., user can configure the PTO axis frequency, closed loop or open loop, encoder channel and average sample.

User can configure the PTO channel as 100Khz or 200Khz. Based on the PTO axis frequency selection, the onboard IO configuration will be updated, and the user need to connect the PTO pulse and Direction sig- nal to the correct hardware channels.

By default, the PTO axis is open loop and however user can also configure a maximum of two PTO axis as closed loop. To configure the axis as closed loop, user need to check the “Closed loop” and select the encoder channel and filter samples if needed. Based on the Encoder Channel selected, user need to connect the encoder signal at the correct hardware channels.

Filter sample will enable a moving average for the output using 2,3 or 4

PTO

PTO frequency 100 kHz ▾

☐ Closed loop Closed loop encoder channel Channel 0 ▾ Average filter samples Off ▾

Unit

Based on the application requirement user can select the desired unit in the wizard and the wizard will update the subsequent parameters to the selected user unit. From the below picture user can find the currently supported unit formats.

Unit

☐ pulse ☐ mm ☐ μm ☐ nm ☒ degree ☐ inch ☐ revolution

Note that for modulo (rotary) axis the units 'mm', 'um', 'nm' and 'inch' might lead to inaccuracy

As an example, when the user selects the axis type as Modulo(rotary) and unit as degree, the wizard will update the subsequent parameters to the selected user unit and fill with default values, ex: modulo range = 360 degree (default). However please make sure the user updates the subsequent parameters as per the actual application requirement.

Note: For rotary axis the units mm, μm, nm, and inch might lead to inaccuracy.

Pulses per revolution scaling

When the axis is open loop, user can update the “Pulses per revolution scaling” with the steps per revolution.

Pulses per revolution scaling

Steps per revolution (1) 2000 steps / revolution

Please note , when the PTO axis is open loop, it is important that the user set the Steps per revolution and Maximum Rpm by keeping the PTO frequency limits into consideration. For example, for an 100Khz PTO axis if the steps per revolution is 2000 , the maximum RPM the axis can support is 3000 RPM. (= $2000 * 3000 / 60 = 100\text{Khz}$)

When the axis is closed loop, user can update the “Pulses per revolution scaling” with the actual encoder increments per motor revolution.

Pulses per revolution scaling

Encoder increments per motor revolution (1) 1048576 pulses/revolution

Application gearing

Based on the actual application requirement, here user can check / uncheck the “Application has gearing” check box. Here the user can also update the required tool travel distance per motor revolution.

When the user unchecks the “Application has gearing”, user can update the “tool travel distance per motor revolution” as per the application requirement as shown in the below picture.

Application gearing

☐ Application has gearing

Units per revolution scaling (without gearbox)

Tool travel distance per motor revolution (2)

360

degree /revolution

Reference: Unit conversion formula

Number of pulses (pulse) =

Encoder increments per revolution (1)

Tool travel distance per motor revolution (2)

* Travel distance (in user defined units)

When the user checks the “Application has gearing”, user will be prompted to provide the gear box details additionally as shown in the below picture and during the generate application, the wizard will update the same accordingly.

Application gearing

☒ Application has gearing

Units per revolution scaling (with gearbox)

Tool travel distance per Gearbox output side revolution (3)

1

degree /revolution

Gearbox output turns: Tooling side (Numerator of reduction ratio) (4)

1

Gearbox input turns: Motor side (Denominator of reduction ratio) (5)

1

Unit conversion formula (modulo)

Number of pulses (pulse) =

Encoder increments per motor revolution (1) * Gear box output turns: Tooling side (4)

Tool travel distance (3) * Gearbox input turns: Motor side (5)

* Travel distance (in user defined units)

Modulo range

The user can provide the modulo range here. This is the value at which the axis position will wrap back to zero. This window will be active only when the user selects the axis type as any of the “rotary” axis.

Modulo range

Modulo range (0-value)

360

degree

Software Limits

The user can configure some of the common “Software Limits” from the wizard itself. Below is the list of software limits which user can configure from the wizard in the selected application units.

By default, software limits in wizard are not enabled and user need to enable the same by enabling the check box “Enable Limits”.

Software limits

☐ Enable limits

Forward limit (axis stop)

1000

degree

Reverse limit (axis stop)

0

degree

Forward limit (warning)

990

degree

Reverse limit (warning)

10

degree

Direction correction

In some of the application we need to change the ax relationship between its real direction and that within the PLC program. By default, the check box will be unchecked, and the direction will be normal. By

selecting the check box “Invert direction” both actual and reference position will be inverted, and the axis will move in opposite direction.

Direction correction	
<input type="checkbox"/>	Invert direction

Position control (cyclic sync mode)

Here the user can configure the parameters related to position control and supervision. Details of each parameter is explained well in our system technology chapter “Position control loop” and “Supervision” under Axis parameter chapter (PLC Automation with V3 CPUs > Libraries and solutions > Motion control library > PLC-based motion control > Axis parameters).

Position control (cyclic sync mode)	
Control time	100 ms
Feed forward percentage (0-100%)	25 %
Following error percentage (0-300%)	200 %
Delay time velocity check	10000 ms
Position lag supervision	Deactivated

Dynamic limits

Users can update the maximum limits here. Some parameters depend on the drive settings and needs to be set correctly to get the desired result.

The user can set the maximum application velocity to a desired value to limit the maximum application speed.

Dynamic limits	
Maximum application velocity	36000 pulses /sec
Maximum acceleration	10000 pulses /sec ²
Maximum deceleration	10000 pulses /sec ²
Maximum jerk	2000 pulses /sec ³

Drive based limits

It is recommended to keep the same Maximum speed at the drive and at the PLC parameter.

Currently the applications torque limits in the wizard are not valid for PTO axis and this is ignored.

Drive based limits	
Maximum speed (user defined)	3000 rpm Maximum speed allowed with this configuration is: 3000

Please note , when the PTO axis is open loop, it is important that the user set the Steps per revolution and Maximum Rpm by keeping the PTO frequency limits into consideration. For example, for an 100Khz PTO axis if the steps per revolution is 2000, the maximum RPM the axis can support is 3000 RPM. (= 2000 * 3000/60 = 100Khz)

When closed loop, please set the Maximum RPM for the axis to reach when the maximum PTO Axis frequency is provided. For example, if the axis is configured as 200Khz and closed loop, set the Maximum RPM for the axis to reach when the 200Khz frequency is achieved.

Results (calculated)

Based on the inputs provided, wizard will calculate the results and can be viewed immediately at the end of the configuration page.

Results (calculated)			
Position resolution	364.0889	pulses/	degree
Maximum possible velocity	36000	degree	/sec
Maximum allowed following error	3600	degree	

8.3.4 Add EtherCAT axis

Limits on number of synchronized EtherCAT axis

Due to differences in the performance of CPU types, there are different limits on:
the minimum EtherCAT cycle time configurable in each PLC type.

PLC Type	PM5630	PM5650	PM5670/PM5675
Min. EtherCAT master cycle time	2 ms	1 ms	0.5 ms

the configurable number of synchronized axis in each PLC type.
These limits are based on the EtherCAT Master cycle time configured in the EtherCAT master.

PLC Type	PM5630	PM5650	PM5670/PM5675
Number of synchronized axis in 1 ms	-	8	16
Number of synchronized axis in 2 ms	4	16	32
Number of synchronized axis in 4 ms	8	32	64

The “Number of axis” is counted in Automation Builder and based on the number of Kernel function block instance declared in the IEC program. therefore also virtual axis are counted.



Note: User can increase the EtherCAT cycle time to accommodate more “number of axis” in the same PLC type.

The “Statistics” tab from Automation Builder can be used to see how many axis are supported and how many are already used for the particular PLC type and for the EtherCAT master cycle time configured. Once the axis are configured/changes done the Statistics tab has to be refreshed by “Generate Code” to get the updated projects information.

The Automation Builder allows one additional axis than what is the catalog or mentioned limit in the above table to account for e.g. one virtual axis additionally.



Note: Please make sure to remove any Kernel function block instance which is declared but not used in the application to get the expected number of axis calculated by Automation Builder under the “Statistics” tab.

After creating the hardware tree, Automation Builder will now pop up “Add Motion Drive” window. Here it shows all the installed EtherCAT drives.

Select the servo drive type you will use, then give a name to the axis and click “Add motion drive” button. This will add the specified servo drive under EtherCAT master module. Each servo drive added under the EtherCAT master will be counted as a motion axis in Automation Builder.

After adding all the drives, click on “Close” button to come out from the pop-up window.

Add Motion Drive

Object path:
PLC_AC500_V3\Extension_Bus\CM579\ETHCAT_Master

Object name: MicroFlex_e190

Categories

ABB - Drives

Search object name...

Name	Short Description	Version	Order Number
MicroFlex e190	MicroFlex e190 Build 5903.4 (CoE)	5.9.03.4	MicroFlex e190
MotiFlex e180	MotiFlex e180 Build 5903.4 (CoE)	5.9.03.4	MotiFlex e180

☐ Close this dialog after each transaction

☐ Display all versions

Reset filter

Add motion drive

Close

Note: If user wants to add multiple servo drives, please uncheck the “*Close this dialog after each transaction*” check box (in left bottom corner) and add multiple drives as per the application requirement in a sequence how it is connected.

If the user wants to add more drives later, the user can always right click on the EtherCAT master object and select add object option which will then pop up the window with supported EtherCAT devices and add the drives as per the requirement.

8.3.4.1 Configuring the CM579-ETHCAT EtherCAT master

This section allows us to configure the behaviour of the Communication module. From here we can define how the hardware will behave. It’s important to understand the EtherCAT master once added is split into two parts in the project tree. These two parts are described below.

CM579-EtherCAT

The label will normally be in the format of ‘[name](CM579-ETHCAT)’ this can be changed by the user but is normally left as default. If so, the name will be **CM579** (CM579-ETHCAT)

In most cases these settings can be left at default but occasionally the user might need to change these to fit the application requirements. Below you can see Run On Config Fault is set to yes meaning the CM579-ETHCAT will not go into error if a slave is lost. Also “Optimize I/O update” is also set to “On” meaning that any EtherCAT PDO I/O will only be updated if used in the code.

CM579				
CM579-ETHCAT Parameters				
CM579-ETHCAT IEC Objects				
Information				
Parameter	Type	Value	Description	
Run on config fault	Enumeration of BYTE	Yes	Start PLC program even on configuration fault	
Broken slave behaviour	Enumeration of DWORD	Leave all broken slaves down	Behaviour of broken slaves	
Distributed clocks	Enumeration of DWORD	Active	Distributed clocks inactive or active	
Bus Target State	Enumeration of BYTE	Operational, OP	Target state of the EtherCAT bus at application	
Bus behavior	Enumeration of DWORD	Asynchronous (IEC bus cycle)	Sync mode 1 - minimum lag (1 bus cycle) between	
Optimize I/O update	Enumeration of BYTE	On	Optimize I/O update	

EtherCAT_Master

The label will normally be in the format of '[name](ETHCAT-Master)' this can be changed by the user but is normally left as default. If so the name will be **ETHCAT_Master** (ETHCAT-Master)

This second part of the EtherCAT master defines the specific settings that define the behavior of the EtherCAT operation.

If the “auto-config master/slaves” mode is activated, then the parameters are set automatically here in accordance with the default settings. This setting is recommended unless the user is very familiar with the setup of EtherCAT networks.

The Default EtherCAT cycle time is 4000 µs but based on the application requirement, user can adapt the EtherCAT cycle time as as shown below.

Please note, EtherCAT cycle time will directly influence the PLC load. If your PLC load is higher than desired, please increase the cycle time or upgrade the PLC type to a higher one.

There are limit on the minimum cycle time for each PLC type and limit on the number of servo drives can be connected for each PLC type based on the cycle time configured. Please refer the help file for more details (PLC Automation with V3 CPUs > Libraries and solutions > Motion control library > Preconditions for the use of the libraries) and adapt the cycle time or PLC Type accordingly to avoid error messages during program download.

User can also check the number of servo axes configured for the given project by checking the slider at the bottom of the Motion Solution Wizard overview page:

8.3.4.2 Configuring the EtherCAT Slave axis

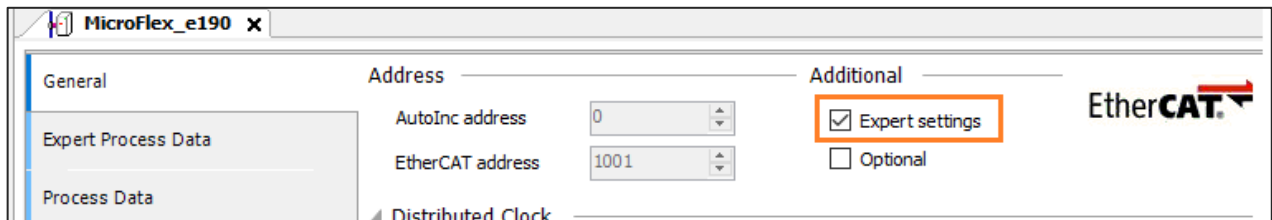
This section allows us to configure the behaviour of the Comms Slave module. From here we can define how the hardware will behave. It's important to understand the EtherCAT Slave Axis once added is split into two parts in the project tree. These two parts are described below.

EtherCAT Slave Object

The label will normally be in the format of '[Drive name](Drive Type)' this can be changed by the user but is normally left as default. If so, the name could be **MicroFlex_e190** (MicroFlex-e190)

In most cases these settings can be left at default but occasionally the user might need to change these to fit the application requirements. Useful settings may be to enable Expert Settings to add additional PDO mappings, to check and set mappings or to check the status of the device once online.

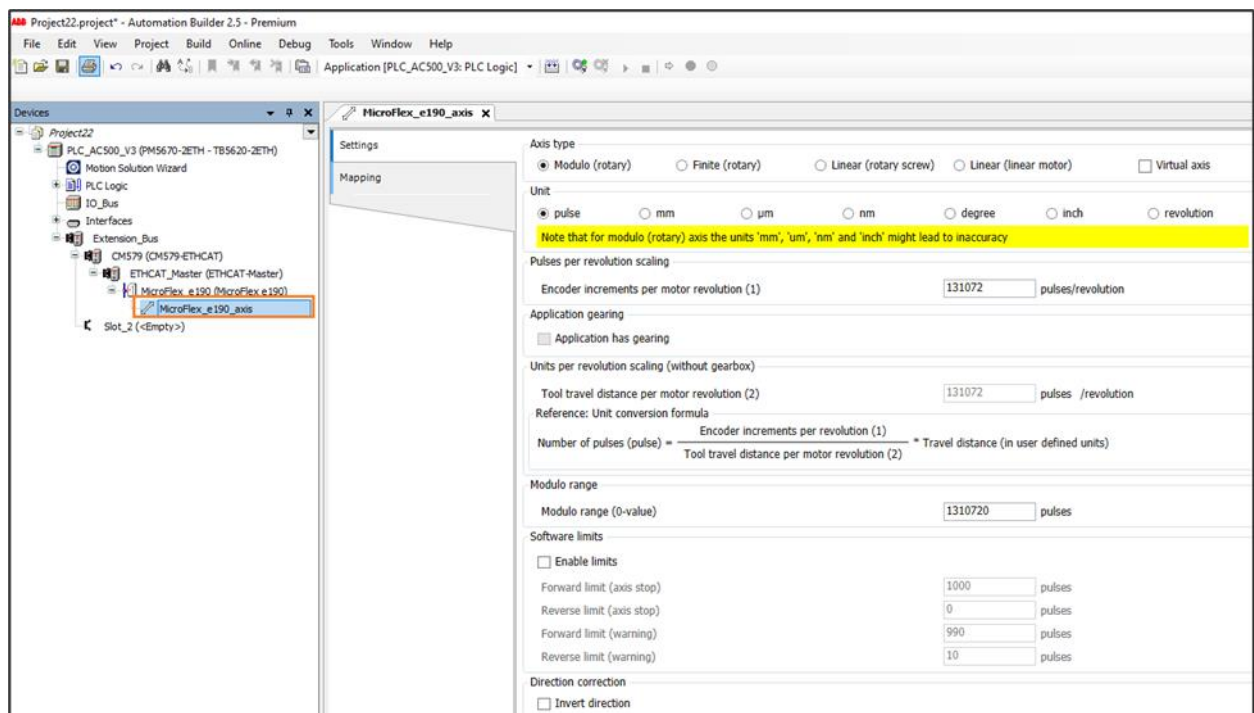
If the user wants to add additional PDO mapped objects First enable the expert settings, then the user will get an additional tab called "Expert Process Data" below the general tab and here user can add / edit / delete the mapping.



EtherCAT Axis Object

Nested underneath the EtherCAT slave object is the Axis object. From here users can configure each axis separately as per the application requirement by opening the motion axis object which is added under the servo drive. To do this double click on the object to open the configuration page which then has two separate tabs, one for "Settings" and the other one for "Mapping". User can update both the pages as per the application requirement.

Later during the "Generate application" process, the wizard will use these settings to define the operation and scaling of this axis object.



EtherCAT Axis Object settings

All settings related to the application and axis specific will be done here and needs to be carefully updated for each axis. Based on the inputs provided here, wizard will compile and generated the code.

Axis Type

The user can select the type of Axis to be configured based on the application requirement. Below is the list of settings which user can configure from the Motion Control wizard along with their meanings.

Axis type

☒ Modulo (rotary)
 ☐ Finite (rotary)
 ☐ Linear (rotary screw)
 ☐ Linear (linear motor)
 ☐ Axis simulation mode

Modulo (rotary)	Default setting in the wizard. By selecting the Modulo (Rotary) your axis will be configured as a roll-over axis and the desired modulo range can be configured later.
Finite (rotary)	Your axis will be configured as a roll-over axis where in modulo range is non editable by the user and calculated based on the “Unit” selection, Inc_Per_R, U_Per_Rev_Nominator and U_Per_Rev_Denominator setting.
Linear (rotary screw)	This needs to be configured when the user having a rotary motor with linear movements (linear axis).
Linear (linear motor)	This needs to be configured when the user axis is a linear motor.
Axis simulation mode	This option is read only from here and needed when the user configured the axis but not have the real hardware yet. This can be selected from the motion solution wizard overview page. This is different than a virtual axis. For virtual axis configuration please refer to the chapter “Add and configure virtual axis”.

Unit

Based on the application requirement user can select the desired unit in the wizard and the wizard will update the subsequent parameters to the selected user unit. From the below picture user can find the currently supported unit formats.

Unit

☐ pulse
 ☐ mm
 ☐ μm
☐ nm
 ☒ degree
 ☐ inch
 ☐ revolution

Note that for modulo (rotary) axis the units 'mm', ' μm ', 'nm' and 'inch' might lead to inaccuracy

As an example, when the user selects the axis type as Modulo(rotary) and unit as degree, the wizard will update the subsequent parameters to the selected user unit and fill with default values, ex: modulo range = 360 degree (default). However please make sure the user updates the subsequent parameters as per the actual application requirement.

Note: For rotary axis the units mm, μm , nm, and inch might lead to inaccuracy.

Pulses per revolution scaling

User can update the “Pulses per revolution scaling” with the actual encoder increments per motor revolution.

Pulses per revolution scaling

Encoder increments per motor revolution (1) 131072 pulses/revolution

Application gearing

Based on the actual application requirement, here user can check / uncheck the “Application has gearing” check box. Here the user can also update the required tool travel distance per motor revolution.

When the user unchecks the “Application has gearing”, user can update the “tool travel distance per motor revolution” as per the application requirement as shown in the below picture.

Application gearing

☐ Application has gearing

Units per revolution scaling (without gearbox)

Tool travel distance per motor revolution (2)

360

degree /revolution

Reference: Unit conversion formula

Number of pulses (pulse) =

Encoder increments per revolution (1)

Tool travel distance per motor revolution (2)

* Travel distance (in user defined units)

When the user checks the “Application has gearing”, user will be prompted to provide the gear box details additionally as shown in the below picture and during the generate application, the wizard will update the same accordingly.

Application gearing

☒ Application has gearing

Units per revolution scaling (with gearbox)

Tool travel distance per Gearbox output side revolution (3)

1

degree /revolution

Gearbox output turns: Tooling side (Numerator of reduction ratio) (4)

1

Gearbox input turns: Motor side (Denominator of reduction ratio) (5)

1

Unit conversion formula (modulo)

Number of pulses (pulse) =

Encoder increments per motor revolution (1) * Gear box output turns: Tooling side (4)

Tool travel distance (3) * Gearbox input turns: Motor side (5)

* Travel distance (in user defined units)

Modulo range

The user can provide the modulo range here. This is the value at which the axis position will wrap back to zero. This window will be active only when the user selects the axis type as any of the “rotary” axis.

Modulo range

Modulo range (0-value)

360

degree

Software Limits

The user can configure some of the common “Software Limits” from the wizard itself. Below is the list of software limits which user can configure from the wizard in the selected application units.

By default, software limits in wizard are not enabled and user need to enable the same by enabling the check box “Enable Limits”.

Software limits

☐ Enable limits

Forward limit (axis stop)

1000

degree

Reverse limit (axis stop)

0

degree

Forward limit (warning)

990

degree

Reverse limit (warning)

10

degree

Direction and Homing Type

Invert Direction: This needs to be set for the application which needs to change the axis relationship between its real direction and that within the PLC program. By default, the check box will be unchecked, and

the direction will be normal. By selecting the check box “Invert direction” both actual and reference position will be inverted, and the axis will move in opposite direction.

Direction and Homing type	
<input type="checkbox"/> Invert direction	<input type="checkbox"/> Homing using DRIVE IO Touch Probe

Homing using Drive IO Touch probe: Selecting this will make the axis to home to a Touch Probe using EtherCAT. Selecting this also by default make the PDO mapping preselected for the user, and this can be later changed based on the application need.

Additional PDO mapping	
<input checked="" type="checkbox"/> Touch probe 1	
<input checked="" type="checkbox"/> Rising Edge (Pos)	
<input type="checkbox"/> Falling Edge (Neg)	
<input type="checkbox"/> Touch probe 2	
<input type="checkbox"/> Rising Edge (Pos)	
<input type="checkbox"/> Falling Edge (Neg)	

Position control (cyclic sync mode)

Here the user can configure the parameters related to position control and supervision. Details of each parameter is explained well in system technology chapter “Position control loop” and “Supervision” under Axis parameter chapter (PLC Automation with V3 CPUs > Libraries and solutions > Motion control library > PLC-based motion control > Axis parameters).

Position control (cyclic sync mode)	
Control time	<input type="text" value="100"/> ms
Feed forward percentage (0-100%)	<input type="text" value="25"/> %
Following error percentage (0-300%)	<input type="text" value="200"/> %
Delay time velocity check	<input type="text" value="10000"/> ms
Position lag supervision	<input type="button" value="Deactivated"/>

Dynamic limits

Users can update the maximum limits here. Some parameters depend on the drive settings and needs to be set correctly to get the desired result.

The user can set the maximum application velocity to a desired value to limit the maximum application speed.

Dynamic limits	
Maximum application velocity	<input type="text" value="36000"/> pulses /sec
Maximum acceleration	<input type="text" value="10000"/> pulses /sec ²
Maximum deceleration	<input type="text" value="10000"/> pulses /sec ²
Maximum jerk	<input type="text" value="2000"/> pulses /sec ³

For example, if user is using an ABB Servo drive with, 131072 Encoder increments per revolution and a Maximum speed is 6000 RPM

$$\begin{aligned}
 \text{Maximum application velocity} &= \text{Max application velocity in RPM} * \text{Tool travel distance per revolution} \\
 &\quad / 60 * \text{Gearbox nominator} / \text{gearbox denominator} \\
 &= 6000 * 360 / 60 * 1 / 1 = 36000 \text{ degree / sec}
 \end{aligned}$$

For easy calculation of parameters user can use the excel “AC500_V3_MotionControl_Startup guide for MC parameterization.xlsx” from example program folder at C:\Users\Public\Documents\AutomationBuilder\Examples\PS5611-Motion\Documentation.

Drive based limits

It is recommended to keep the same Maximum speed at the drive and at the PLC parameter.

Currently users can define the applications torque limits in the wizard and they will be written to the SDO startup parameter only if the user selects “Torque limits” in “Mapping” page. These parameters are not used in the program by default.

Drive based limits		
Maximum speed (user defined)	<input type="text" value="6000"/>	rpm
Maximum positive torque	<input type="text" value="300"/>	%
Maximum negative torque	<input type="text" value="-300"/>	%

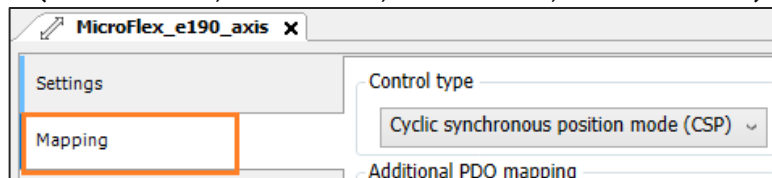
Results (calculated)

Based on the inputs provided, wizard will calculate the results and can be viewed immediately at the end of the configuration page.

Results (calculated)		
Position resolution	364.0889	pulses/ degree
Maximum possible velocity	36000	degree /sec
Maximum allowed following error	3600	degree

Axis control type and object mapping

The Mapping and Control Type tab can be selected if the user wants to set a Control Type other than CSP (Default) and mappings other than default (Control Word, Set Position, Status Word, Actual Position). It



can be found under the axis object here:

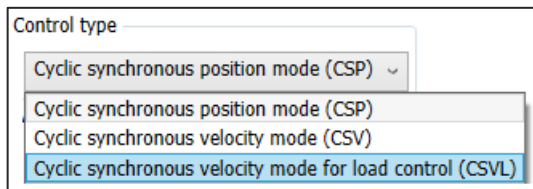
Control type

By default, wizard is selected for cyclic synchronous position mode (CSP). User can change the same based on the application requirement. Currently we support below control modes,

Cyclic synchronous position mode (CSP).

Cyclic synchronous velocity mode (CSV).

Cyclic synchronous velocity mode for load control (CSVL).



Note: CSVL is an ABB specific mode to achieve load control / profiling. By using this mode, the user can use the Motion Control Load library which is implemented based on the “PLCOpen Motion Part 6 – Fluid Power Extensions”. For more details on load / torque control please refer to the library integrated documentation, system technology in online help file and the example program / description from example program folder.

8.3.4.3 Additional PDO mapping

If the application needs additional PDO mapping the wizard helps the user to add most used PDO mapping just by selecting them here.

Based on the control type selected, a few of the mandatory PDO mapping are generated automatically and from additional mapping area in wizard user can find the most common PDO mapping and user can add the same based on the application requirement.

Additional PDO mapping

☐ Touch probe 1

☐ Rising Edge (Pos)
 ☐ Falling Edge (Neg)

☐ Touch probe 2

☐ Rising Edge (Pos)
 ☐ Falling Edge (Neg)

☐ Master encoder
 ☐ Following error
 ☐ Digital input states
 ☐ Digital output force
 ☐ CST Support

☐ Max Profile Velocity
 ☐ Target Torque

SDO startup parameter mapping

☒ Give EtherCAT control
 ☒ Operating mode
 ☐ Torque limits

Please note, the user can add additional PDO mapping which are not listed here manually by enabling the expert settings from the slave device general configuration page (as described earlier).

SDO start-up parameter mapping

By default, two of the SDO startup parameters are selected and it is recommended not to change these unless the user has expert level knowledge of DS 402 control modes or intends to do none standard start up coding as it will change the expected operation of the axis at start up.

SDO startup parameter mapping

☒ Give EtherCAT control
 ☒ Operating mode
 ☐ Torque limits

The user can select the Torque limits and the torque values set from the “settings” page will be written to the respective slave drives startup parameters list.

Drive based limits

Maximum speed (user defined)

6000

rpm

Maximum positive torque

300

%

Maximum negative torque

-300

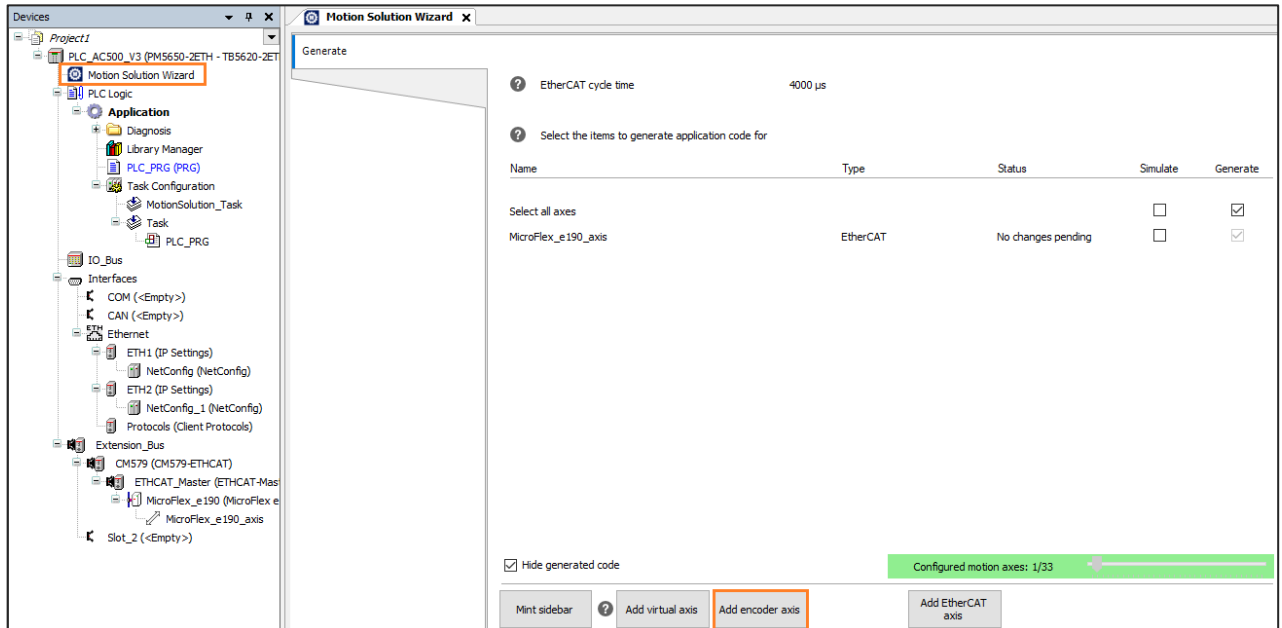
%

Once these settings are made and the Generate code is executed you can see that these settings have change the drives EtherCAT Slave configuration as shown in the picture below:

MicroFlex_e190									
General									
Process Data									
Line	Index/Subindex	Name	Value	Bit Length	Abort on Error	Jump to Line on Error	Next Line	Comment	
1	16#5002:16#00	Give EtherCAT Control	1	16	<input type="checkbox"/>	<input type="checkbox"/>	0	Give EtherCAT Control	
2	16#6060:16#00	ModesOfOperation = 8 (Cyclic Synchronous Position Mode)	8	8	<input type="checkbox"/>	<input type="checkbox"/>	0	ModesOfOperation = 8 (Cyclic Synch	
3	16#5023:16#00	Maximum Torque Limit	3000	16	<input type="checkbox"/>	<input type="checkbox"/>	0	Maximum Torque Limit	
4	16#5022:16#00	Minimum Torque Limit	-3000	16	<input type="checkbox"/>	<input type="checkbox"/>	0	Minimum Torque Limit	

8.3.5 Adding encoder axis

The application which needs an encoder axis to be configured, users can add encoder axis under “Motion Solution Wizard” object either by open the “Motion Solution Wizard” object by doubling clicking on the same and clicking on the “Add encoder axis” button on the bottom of the page as shown below or by right clicking on the “Motion Solution Wizard” and select “Add object” option.

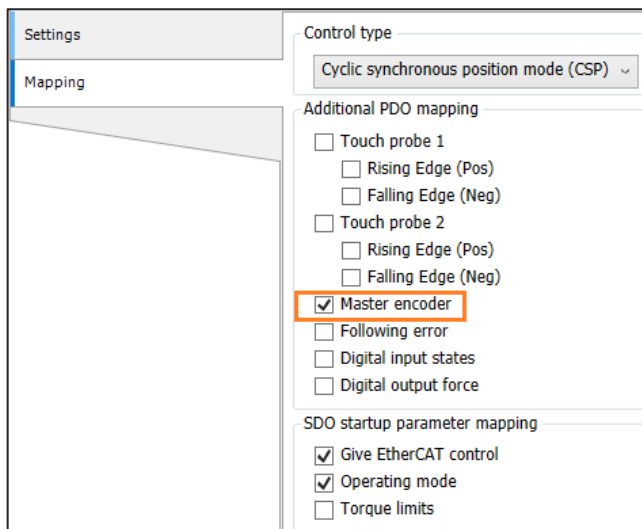


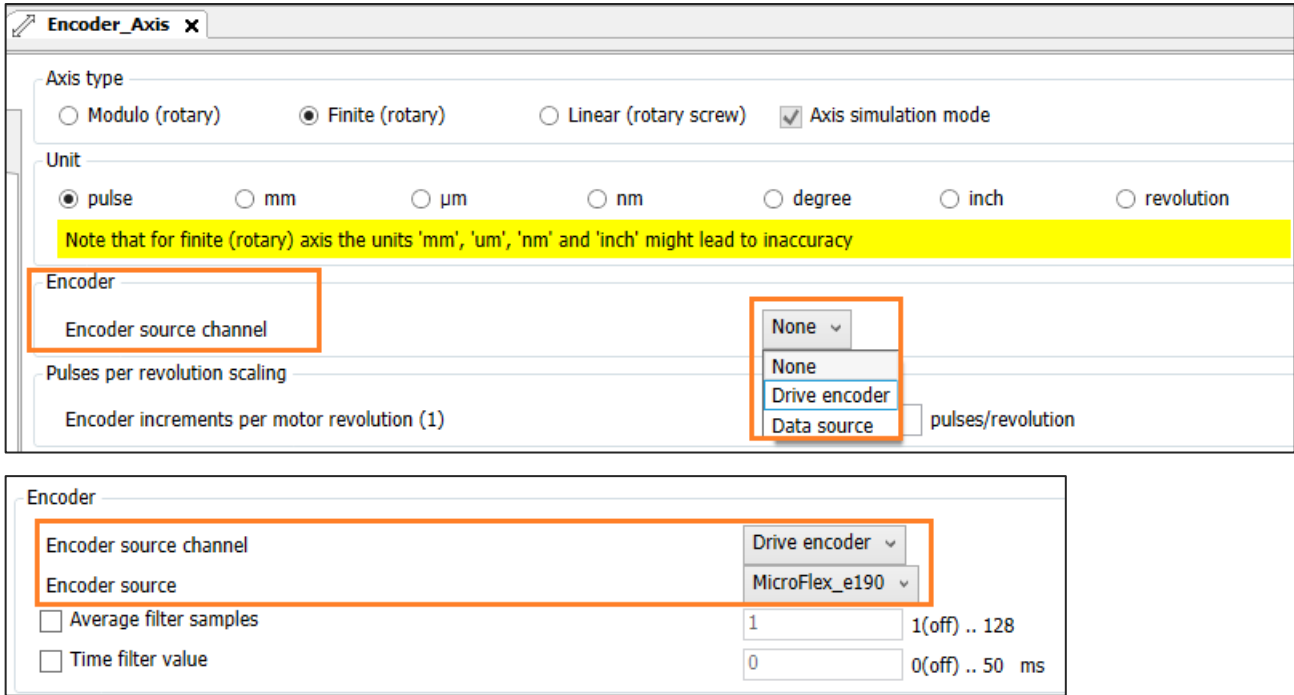
After adding the encoder axis, user can find the same object under “Motion Solution Wizard” in Automation Builder. Users can double click on the added encoder axis object to get the settings page and configure it as per the requirement. Settings here is like any other motion axis expect here user need to configure encoder source.

When an Encoder Axis is added, user must configure the Encoder source channel and Encoder Source.

8.3.5.1 Drive Encoder

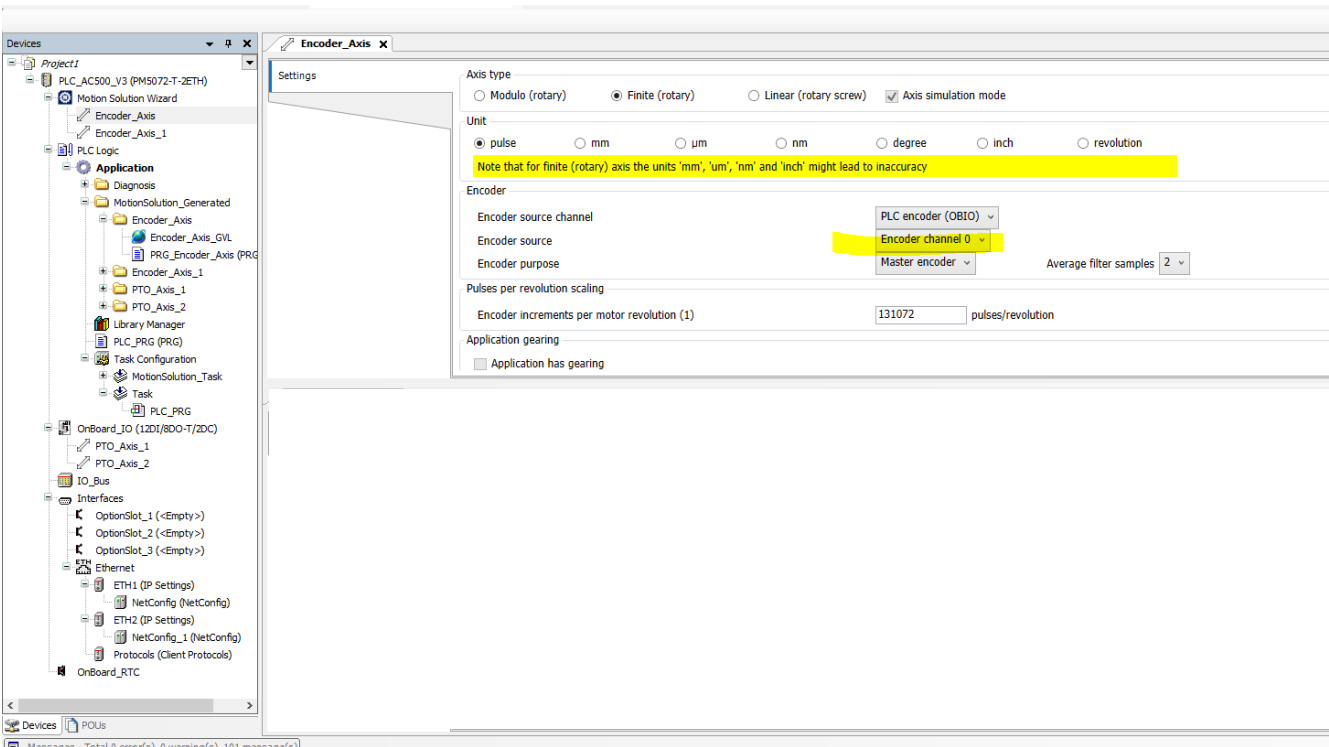
When there is an EtherCAT axis, user will have a chance to use the EtherCAT Servo Drives master encoder (16#400C) as a source for the Encoder axis. For this, user must have an EtherCAT axis configured and “Master encoder” PDO mapping is checked.





8.3.5.2 PLC Encoder (OBIO)

When there is an PTO axis, user will have an option to use the onboard encoder channels from eCo PLC as a source for the Encoder axis.



8.3.5.3 Data Source

User can use the “Data source” and define a variable which will then create the variable in the mentioned name as data type “DINT”. User must make sure the encoder value is feed into the generated or mapped variable in the project.



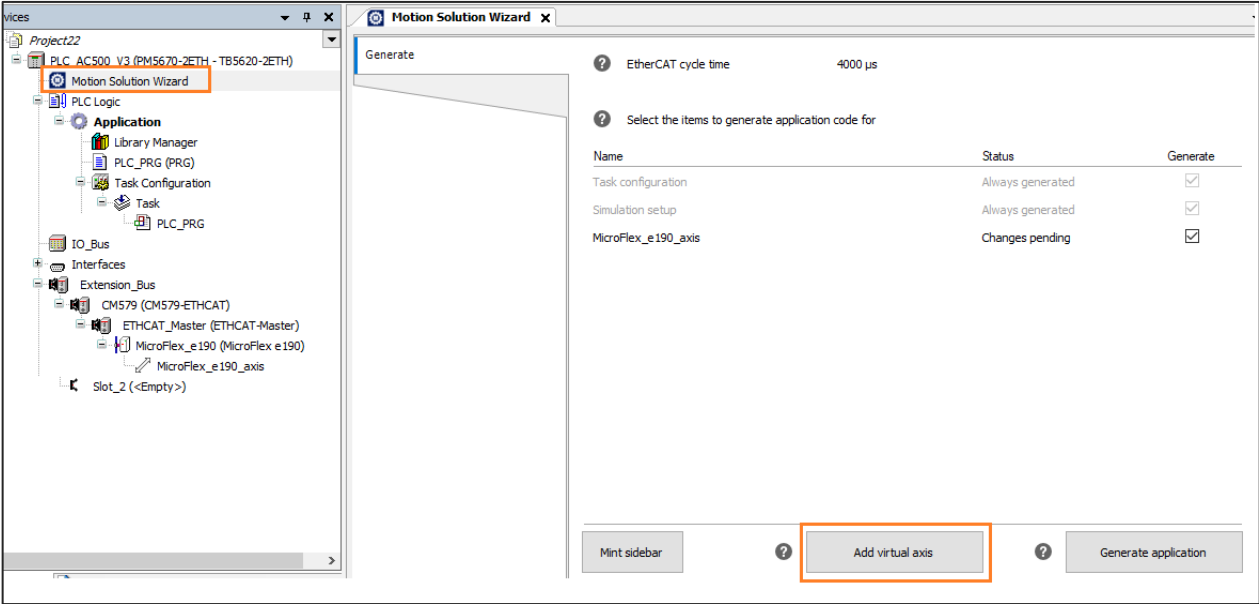
or connect an existing variable from the project to Encoder axis. Please make sure the variable which is connect is type “DINT”



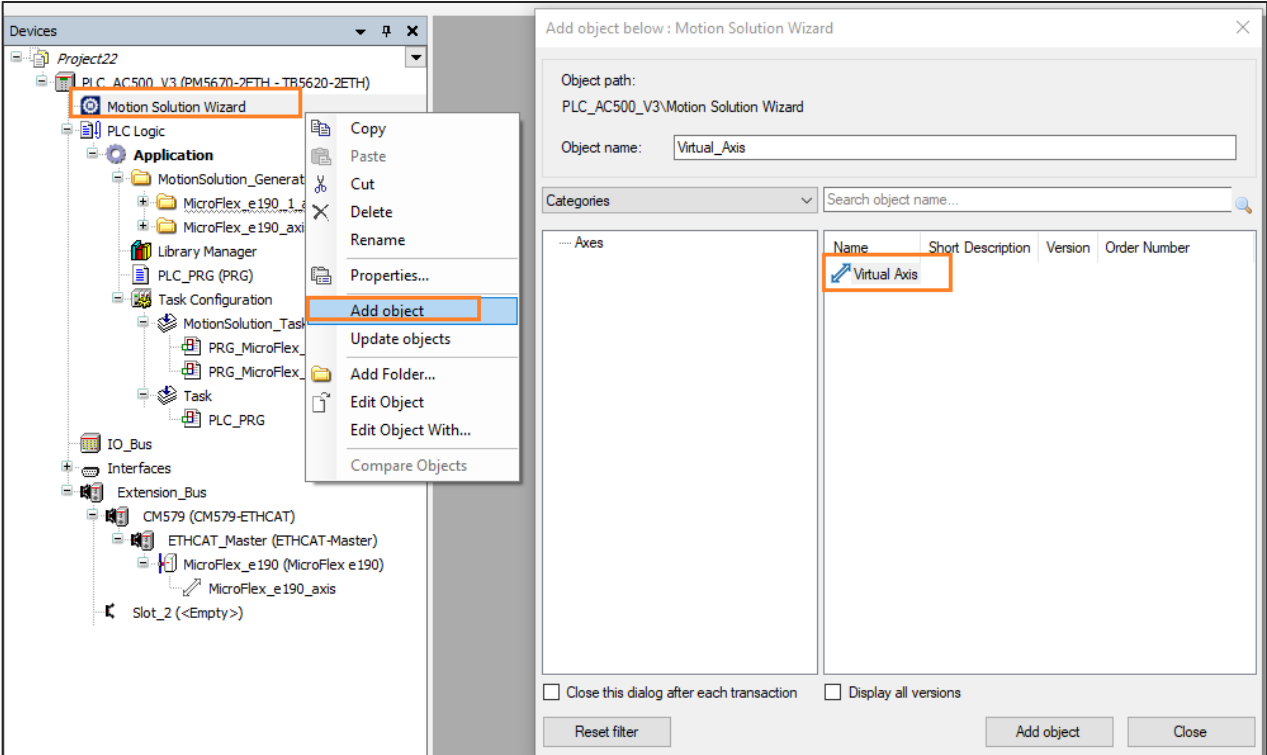
Other Encoder Axis parameters are same as EtherCAT or PTO axis and for details please refer to the previous chapters.

8.3.6 Adding virtual axis

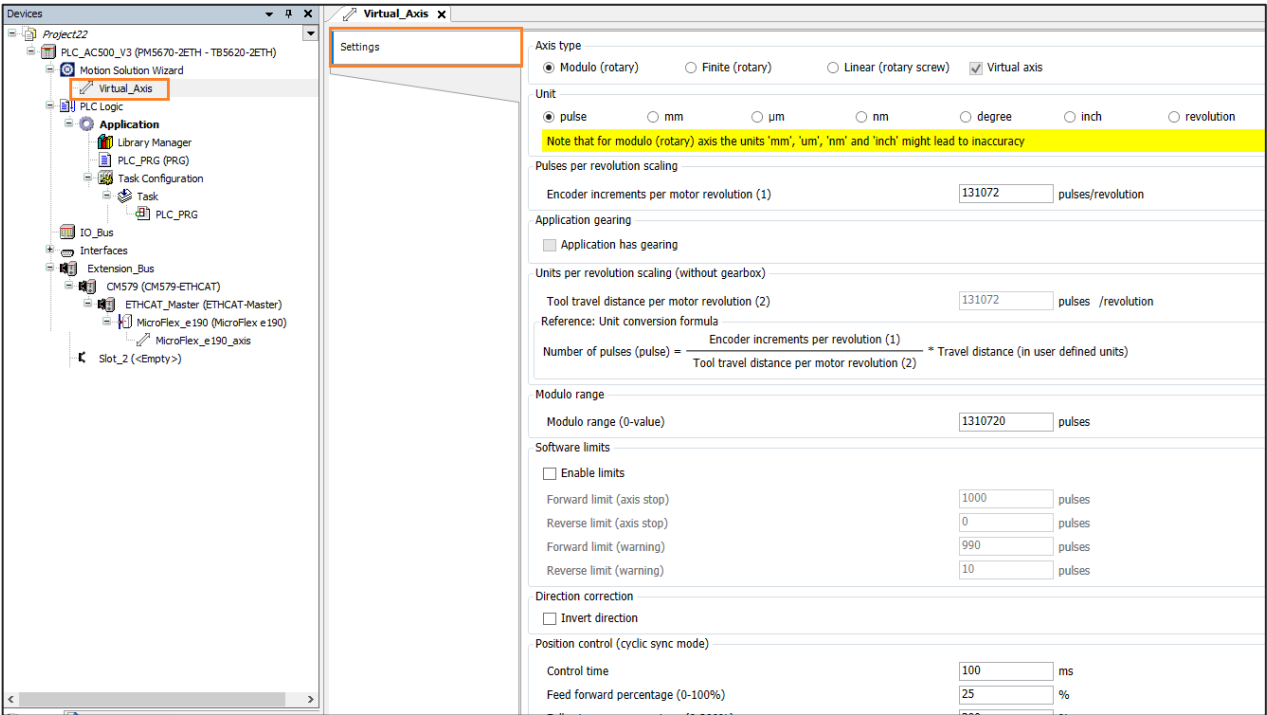
The applications which needs an virtual axis , users can add virtual axis under “Motion Solution Wizard” object either by open the “Motion Solution Wizard” object by doubling clicking on the same and clicking on the “Add virtual axis” button on the bottom of the page as shown below.



or by right clicking on the “Motion Solution Wizard” object and by selecting add object option as shown below.



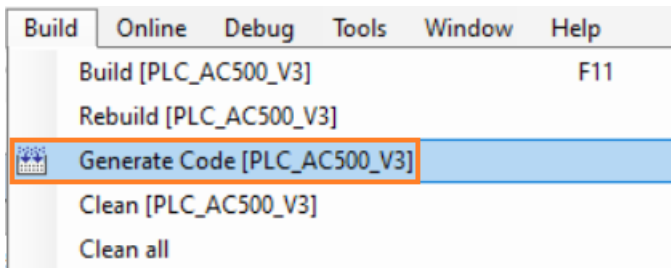
After adding the virtual axis, user can find the same object under “Motion Axis Solution Wizard” in Automation Builder. Users can double click on the added virtual axis object to get the settings page and configure it as per the requirement. Settings here is like the motion axis and details for how to configure the parameters, please refer to the previous chapters.



8.4 Motion Axis generation

Once all the configuration is done, user can generate the application which will then update its settings, generate PDO/SDO mapping, motion task configuration for user application automatically based on the settings and parameters provided to the wizard.

To do this action, go to Build menu and click on Generate Code



Wait for few seconds until Automation Builder will generate the axis configuration based on the parametrization done by the user. Once completed successfully, Automation Builder will generate the message “Motion Solution Generation successful” in the message window. This can take some time based on the number of axis configured in the project and PDO mapping selection.

8.4.1 PTO axis

8.4.1.1 PTO axis onboard IO (OBIO) configuration

Based on the PTO axis frequency configured, the motion solution wizard configures the onboard IO outputs and assign the variables to the output channels. The configuration is done automatically by the wizard based on the axis configuration in Automation Builder.

The axes are counted from axis 1 to 4 based on the order it is configured on the Automation Builder device tree. User can change the axis configured order in Automation Builder by renaming the axis name and during “Generate code” the new configuration will be updated, and user need to connect the pulse and direction cables accordingly.

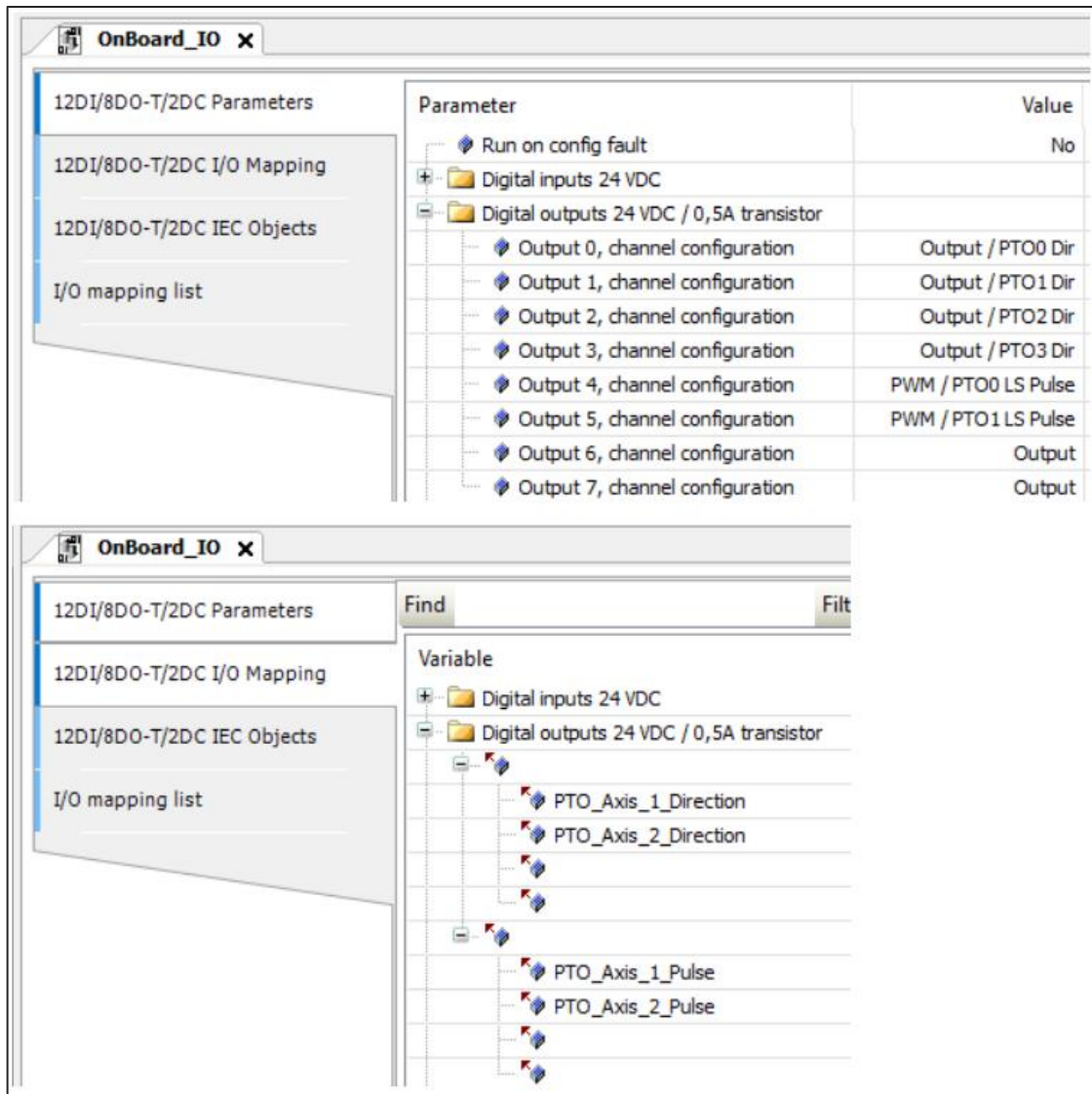
100Khz (default) PTO axis

When the axis is configured as 100Khz, output channel O0 to O3 (Value = PTOx Dir) are configured as direction output and O4 to O7 (Value = PTOx LS Pulse) are configured as pulse. Users need to connect the PTI drive cables accordingly.

Below table shows the user an overview of the hardware channels configured based on the number of axes configured, this can be used as a reference for the pulse and direction wiring to PTI drive.

		Axis frequency in Khz				
		Axis1	100	100	100	100
		Axis2		100	100	100
		Axis3			100	100
		Axis4				100
HW Channel Selection	DO0	PWM/PTO0 Dir	PWM/PTO0 Dir	PWM/PTO0 Dir	PWM/PTO0 Dir	PWM/PTO0 Dir
	DO1	PWM/PTO1 Dir		PWM/PTO1 Dir	PWM/PTO1 Dir	PWM/PTO1 Dir
	DO2	PWM/PTO2 Dir			PWM/PTO2 Dir	PWM/PTO2 Dir
	DO3	PWM/PTO3 Dir				PWM/PTO3 Dir
	DO4	PWM/PTO0 LS Pulse PTO0 HS Pulse / Cw	PWM/PTO0 LS Pulse	PWM/PTO0 LS Pulse	PWM/PTO0 LS Pulse	PWM/PTO0 LS Pulse
	DO5	PWM/PTO1 LS Pulse PTO0 HS Dir / Ccw		PWM/PTO1 LS Pulse	PWM/PTO1 LS Pulse	PWM/PTO1 LS Pulse
	DO6	PWM/PTO2 LS Pulse PTO1 HS Pulse / Cw			PWM/PTO2 LS Pulse	PWM/PTO2 LS Pulse
	DO7	PWM/PTO3 LS Pulse PTO1 HS Dir / Ccw				PWM/PTO3 LS Pulse

For example, when two PTO axes are configured as 100Khz, the motion solution wizard set the onboard output configuration. For the first axis, direction as Output0 and pulse as Output4 and for the second axis direction as Output1 and pulse as Output5.



200Khz PTO axis

When the axis is configured as 200Khz, output channel O4 (Value = PTOx HS Pulse) and O5 (Value = PTOx HS Dir) are configured as pulse and direction output for the first axis and O6 and O7 is configured as pulse and direction output for the second axis.

Below table shows the user an overview of the hardware channels configured based on the number of axes configured, this can be used as a reference for the pulse and direction wiring to PTI drive.

		Axis frequency in Khz	
	Axis1	200	200
	Axis2		200
	Axis3		
	Axis4		
HW Channel Selection	DO0	PWM/PTO0 Dir	
	DO1	PWM/PTO1 Dir	
	DO2	PWM/PTO2 Dir	
	DO3	PWM/PTO3 Dir	
	DO4	PWM/PTO0 LS Pulse PTO0 HS Pulse / Cw	PTO0 HS Pulse / Cw
	DO5	PWM/PTO1 LS Pulse PTO0 HS Dir / Ccw	PTO0 HS Dir / Ccw
	DO6	PWM/PTO2 LS Pulse PTO1 HS Pulse / Cw	PTO1 HS Pulse / Cw
	DO7	PWM/PTO3 LS Pulse PTO1 HS Dir / Ccw	PTO1 HS Dir / Ccw

For example, when two PTO axes are configured as 200Khz, the motion solution wizard set the onboard output configuration. For the first axis, Pulse as Output4 and direction as Output5 and for the second axis pulse as Output6 and direction as Output7.

Parameter	Value
Run on config fault	No
<div> <div>+</div> <div>Digital inputs 24 VDC</div> </div>	
<div> <div>-</div> <div>Digital outputs 24 VDC / 0,5A transistor</div> </div>	
Output 0, channel configuration	Output / PTO0 Dir
Output 1, channel configuration	Output / PTO1 Dir
Output 2, channel configuration	Output / PTO2 Dir
Output 3, channel configuration	Output / PTO3 Dir
Output 4, channel configuration	PTO0 HS Pulse/Cw
Output 5, channel configuration	PTO0 HS Dir/Ccw
Output 6, channel configuration	PTO1 HS Pulse/Cw
Output 7, channel configuration	PTO1 HS Dir/Ccw

Variable	Channel
<div> <div>+</div> <div>Digital inputs 24 VDC</div> </div>	
<div> <div>-</div> <div>Digital outputs 24 VDC / 0,5A transistor</div> </div>	
<div> <div>+</div> <div>PTO_Axis_1_Pulse</div> </div>	Fast outputs DO0-D...
<div> <div>-</div> <div>PTO_Axis_1_Direction</div> </div>	Fast outputs DO4-D...
<div> <div>+</div> <div>PTO_Axis_2_Pulse</div> </div>	Digital output DO4
<div> <div>-</div> <div>PTO_Axis_2_Direction</div> </div>	Digital output DO5
<div> <div>+</div> <div>PTO_Axis_1_Pulse</div> </div>	Digital output DO6
<div> <div>-</div> <div>PTO_Axis_1_Direction</div> </div>	Digital output DO7

100Khz and 200Khz PTO axis

When some axes are configured as 100Khz and some are configured as 200Khz frequency, user must take care following points,

Make sure the 200Khz axis is configured as first or last axis and not in between 100Khz axis.

When the first axis is 200Khz, Automation Builder will configure the O4 and O5 channels and for 100Khz O6 and O7 is configured as Pulse and O2 & O3 are configured as direction.

Below table shows the user an overview of the hardware channels configured based on the number of axes configured.

		Axis frequency in Khz					
		Axis1	100	200	100	200	100
		Axis2	200	100	100	100	200
		Axis3			200	100	100
		Axis4					
HW Channel Selection	DO0	PWM/PTO0 Dir	PWM/PTO0 Dir		PWM/PTO0 Dir		
	DO1	PWM/PTO1 Dir			PWM/PTO1 Dir		
	DO2	PWM/PTO2 Dir		PWM/PTO2 Dir		PWM/PTO2 Dir	
	DO3	PWM/PTO3 Dir				PWM/PTO3 Dir	
	DO4	PWM/PTO0 LS Pulse PTO0 HS Pulse / Cw	PWM/PTO0 LS Pulse	PTO0 HS Pulse / Cw	PWM/PTO0 LS Pulse	PTO0 HS Pulse / Cw	Configuration not allowed. Please keep 200Khz axis as first or last axis
	DO5	PWM/PTO1 LS Pulse PTO0 HS Dir / Ccw		PTO0 HS Dir / Ccw	PWM/PTO1 LS Pulse	PTO0 HS Dir / Ccw	
	DO6	PWM/PTO2 LS Pulse PTO1 HS Pulse / Cw	PTO1 HS Pulse / Cw	PWM/PTO2 LS Pulse	PTO1 HS Pulse / Cw	PWM/PTO2 LS Pulse	
	DO7	PWM/PTO3 LS Pulse PTO1 HS Dir / Ccw	PTO1 HS Dir / Ccw		PTO1 HS Dir / Ccw	PWM/PTO3 LS Pulse	

8.4.1.2 General PLC configuration changes

Compare with standard AC500 project, when the user is using the motion solution wizard, a couple of default settings are changed during Generate code. These settings are always overwritten as long as there is a change in axis configuration or added a new axis. Below are the settings which are updated by motion solution wizard,

1. IO bus – Run on config fault to “Yes”
2. Always update variable – Enabled 1

It is recommended that user change the above settings manually based on the actual application requirement. To update these setting manually, user need to change the setting after all the axis changes are done and at least once Generate Code is executed.

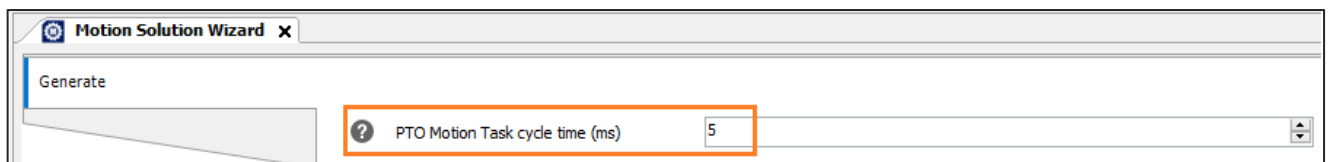
8.4.1.3 Task configuration

Motion wizard will generate a task configuration to run the PTO motion axis. The cycle time for the task needs to be specified at Motion Solution Wizard overview page.

The Motion Solution Wizard will automatically configure all the axis configuration function blocks, and call them from within the task, “MotionSolution_Task” . It is possible to call parts of the motion application in the same “MotionSolution_Task” task - however it is recommended to use a different **cyclic task** for the PLCopen Motion function blocks with a cycle time **double or higher** than MotionSolution_Task. This will reserve more capacities for better performance on the EtherCAT by the PLC.

Please note that some PLCopen function blocks must be called in the MotionSolution_Task.

For non-real time motion parts of the program, it is advised that this part of the program can be called in a separate cyclic task with a lower priority and a longer cycle time. For example, if some code referring to the HMI variables update is needed, it is recommended to set longer cycle time such as 200ms.



It is recommended to use separate cyclic task for customer application / PLC open function blocks since this will reduce unnecessary load.

8.4.2 EtherCAT motion axis

8.4.2.1 General PLC configuration changes

Compare with standard AC500 project, when the user is using motion solution wizard, couple of default settings are changed during Generate code. These settings are always overwritten as long as there is a change in axis configuration or added a new axis. Below are the settings which are updated by motion solution wizard,

1. IO bus – Run on config fault to “Yes”
2. CM579-ETHERCAT - Run on config fault to “Yes”
3. CPU-Parameters – Check battery to “Off”
4. Always update variable – Enabled 1

It is recommended that user change the above settings manually based on the actual application requirement. To update these setting manually, user need to change the setting after all the axis changes are done and at least once Generate Code is executed.

8.4.2.2 Task configuration

Motion wizard will generate a task configuration which is synchronized with the EtherCAT cycle time configured in the EtherCAT master and the Priority is set as “0”.

The Motion Solution Wizard will automatically configure all the axis configuration function blocks, and call them from within the task, “MotionSolution_Task” . It is possible to call parts of the motion application in the same “MotionSolution_Task” task - however it is recommended to use a different **cyclic task** for the PLCopen Motion function blocks with a cycle time **double or higher** than MotionSolution_Task. This will reserve more capacities for better performance on the EtherCAT by the PLC.

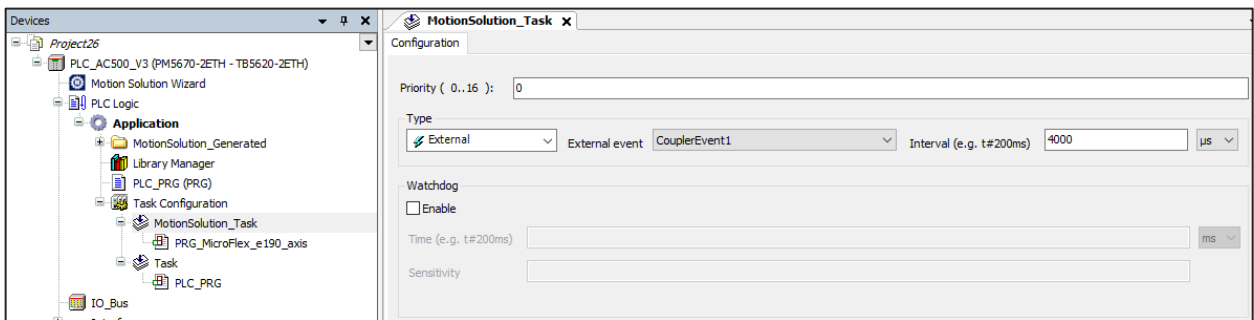
Please note that some PLCopen function blocks must be called in the MotionSolution_Task.

For non-real time motion parts of the program, it is advised that this part of the program can be called in a separate cyclic task with a lower priority and a longer cycle time. For example, if some code referring to the HMI variables update is needed, it is recommended to set longer cycle time such as 200ms.

It is recommended to use separate cyclic task for customer application / PLC open function blocks since this will reduce unnecessary load, especially in the larger applications with many axis.

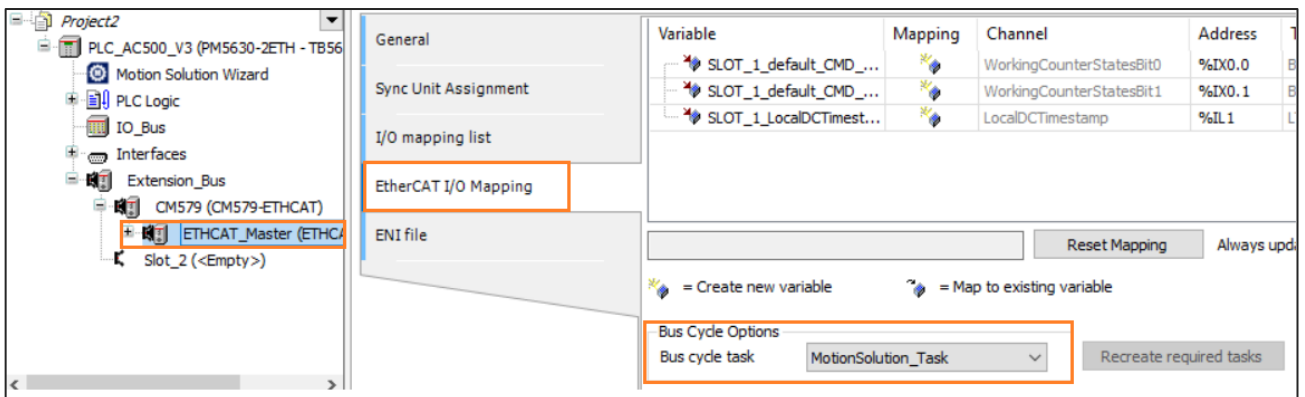


Note: PLC tasks must set a higher watchdog time if the PLC is stopping due to an exception error.

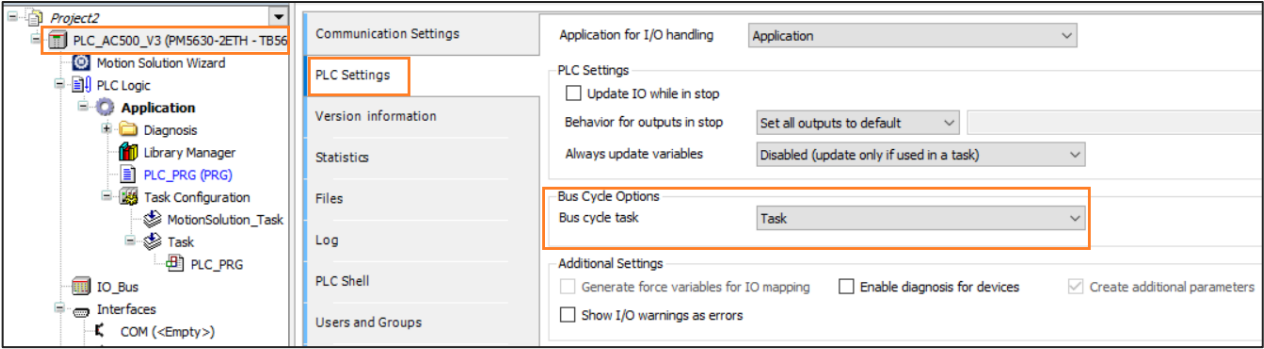


Note: It is recommended not to update the task configuration here manually since this will be lost next time when the user click on the” Generate Code”.

The Motion wizard also set the EtherCAT IO Mapping to the Motion Solution task which is synchronized with the EtherCAT cycle.

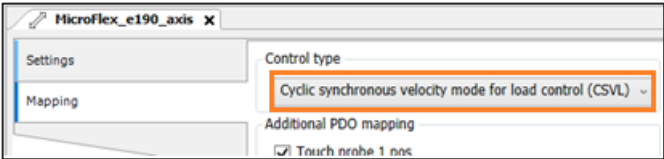


It is highly recommended to set a task for Bus cycle task rather than keeping default setting <un-specified>. Recommended to set a non-motion task for better overall PLC performance.



8.4.2.3 Motion solution libraries

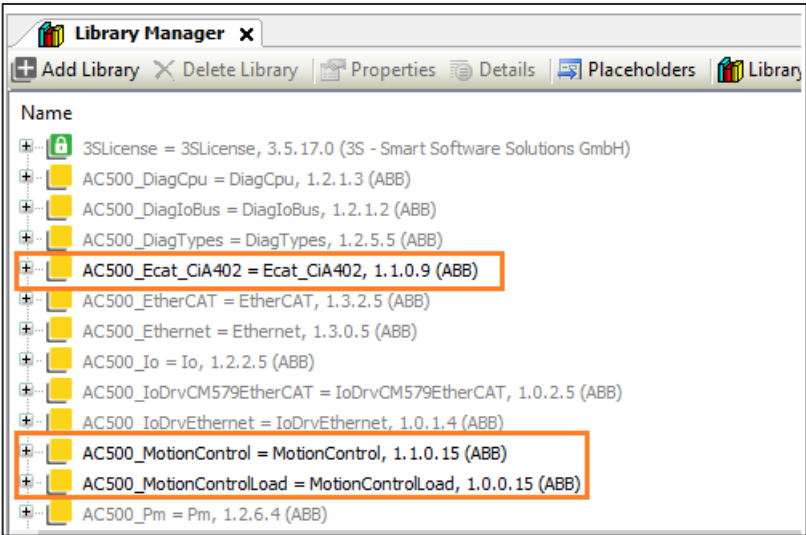
Based on the control type selection under motion axis -> mapping, mandatory libraries will be added to the library manager. The libraries added depend on the Control Type Selected



See the table below for the libraries added dependent on Control Type selected

Control Type	AC500_Ecat_CiA402	AC500_MotionControl	AC500_MotionControlLoad
CSP	Added	Added	
CSV	Added	Added	
CSVL	Added	Added	Added

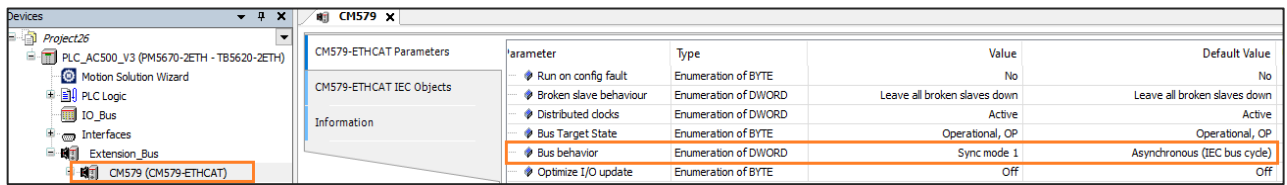
For example, when the user selects control type as CSVL we can see the below libraries indicated in the library manager,



Note: additional libraries can always be added later manually.

8.4.2.4 EtherCAT bus behaviour

The wizard will update the EtherCAT bus behavior in CM579 module parameter setting by updating the Asynchronous (default value) to Sync mode 1.

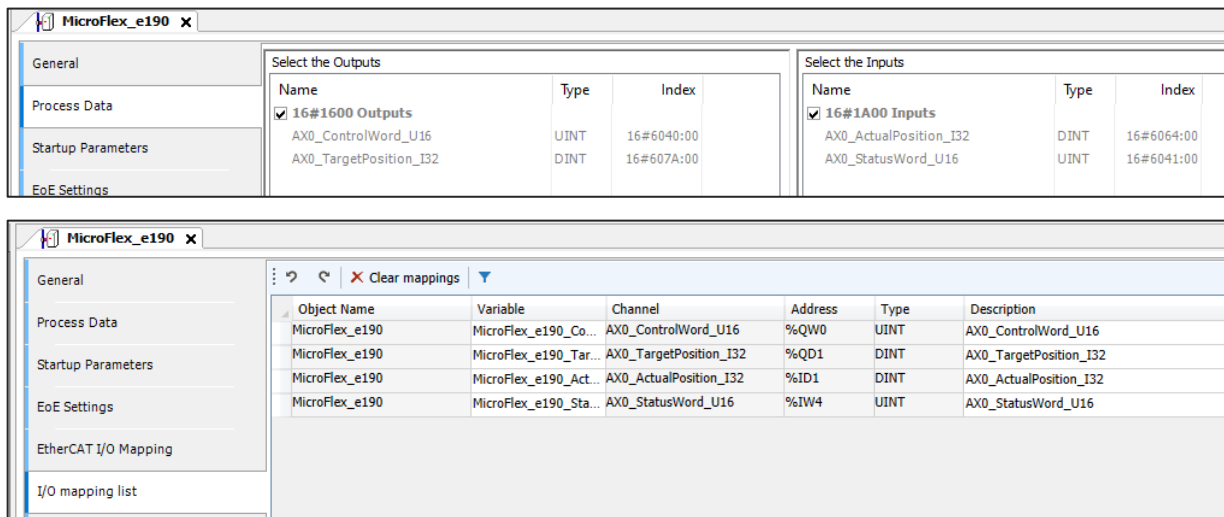


Parameter	Type	Value	Default Value
Run on config fault	Enumeration of BYTE	No	No
Broken slave behaviour	Enumeration of DWORD	Leave all broken slaves down	Leave all broken slaves down
Distributed clocks	Enumeration of DWORD	Active	Active
Bus Target State	Enumeration of BYTE	Operational, OP	Operational, OP
Bus behavior	Enumeration of DWORD	Sync mode 1	Asynchronous (IEC bus cycle)
Optimize I/O update	Enumeration of BYTE	Off	Off

8.4.2.5 PDO and Startup Parameters (SDO)

Based on the selection under motion axis -> mapping, PDO and startup parameters are generated automatically as shown below.

Based on the control type and PDO mapping selected in the wizard, it will update the Process Data tab from the slave axis object and assign the autogenerated name to each object which is added.

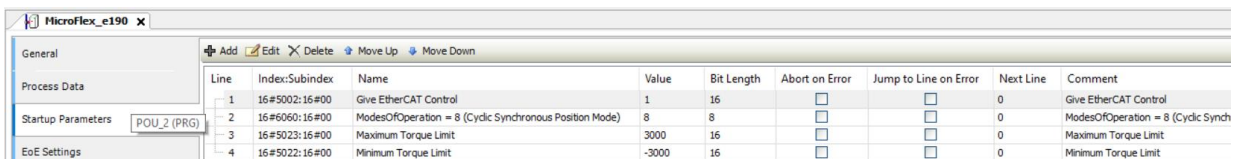


Name	Type	Index
<input checked="" type="checkbox"/> 16#1600 Outputs		
AX0_ControlWord_U16	UINT	16#6040:00
AX0_TargetPosition_I32	DINT	16#607A:00

Name	Type	Index
<input checked="" type="checkbox"/> 16#1A00 Inputs		
AX0_ActualPosition_I32	DINT	16#6064:00
AX0_StatusWord_U16	UINT	16#6041:00

Object Name	Variable	Channel	Address	Type	Description
MicroFlex_e190	MicroFlex_e190_Co...	AX0_ControlWord_U16	%QW0	UINT	AX0_ControlWord_U16
MicroFlex_e190	MicroFlex_e190_Tar...	AX0_TargetPosition_I32	%QD1	DINT	AX0_TargetPosition_I32
MicroFlex_e190	MicroFlex_e190_Act...	AX0_ActualPosition_I32	%ID1	DINT	AX0_ActualPosition_I32
MicroFlex_e190	MicroFlex_e190_Sta...	AX0_StatusWord_U16	%IW4	UINT	AX0_StatusWord_U16

Like PDO mapping, SDO startup will be updated based on the control type and the mapping selected in the wizard.



Line	Index/Subindex	Name	Value	Bit Length	Abort on Error	Jump to Line on Error	Next Line	Comment
1	16#5002:16#00	Give EtherCAT Control	1	16	<input type="checkbox"/>	<input type="checkbox"/>	0	Give EtherCAT Control
2	16#6060:16#00	ModesOfOperation = 8 (Cyclic Synchronous Position Mode)	8	8	<input type="checkbox"/>	<input type="checkbox"/>	0	ModesOfOperation = 8 (Cyclic Synch
3	16#5023:16#00	Maximum Torque Limit	3000	16	<input type="checkbox"/>	<input type="checkbox"/>	0	Maximum Torque Limit
4	16#5022:16#00	Minimum Torque Limit	-3000	16	<input type="checkbox"/>	<input type="checkbox"/>	0	Minimum Torque Limit



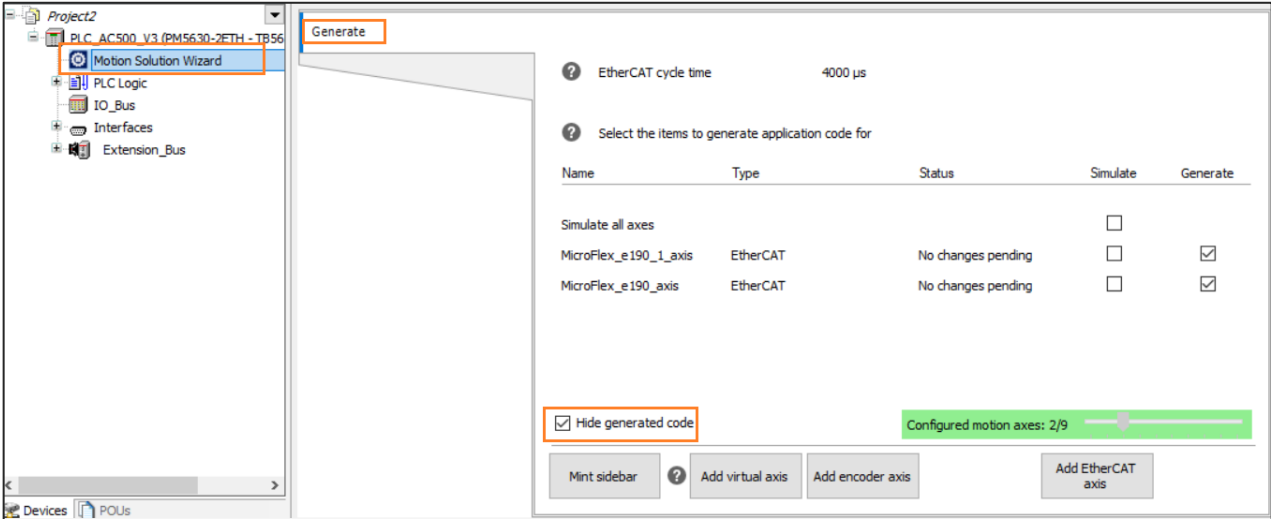
Note: Based on the application requirement, user can add more PDO / startup parameters manually.

8.4.3 Axis program generated (Hidden by default)

The axis programs which are generated for each axis is hidden by default but for expert users can modify the same if needed.

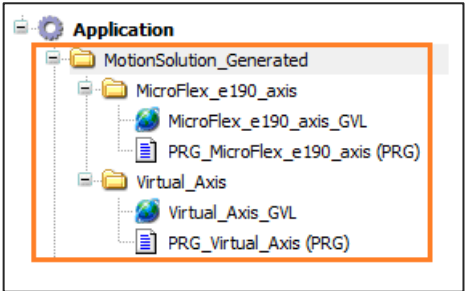
It is recommended not to change the program manually if not an expert user and any changes which will be done later the axis object will overwrite all the changes done by the user if the "Generate" check box is checked.

To view the generated axis program, user need to uncheck the “Hide generated code” from motion solution wizard overview page.



Once unchecked the “Hide generated code”, user can find the folder name” MotionSolution_Generated”, which has sub folders for each of the axis configured using the wizard. Each subfolder is having a GVL and a PRG respectively. Name of the folder / GVL / PRG / Function Blocks / variable are generated based on the axis name provided.

Below is the example from EtherCAT axis, for other axis it follows the similar structure.



8.4.3.1 Axis parameters generated (GVL)

GVL which is generated by the wizard maps the parameters set from the wizard with the library variables in the Axis_Ref structure and the Axis_IO declaration is initiated here.


```


2  VAR_GLOBAL
3      MicroFlex_e190_Kernel : CMC_Basic_Kernel;  (*Axis reference for MicroFlex_e190_Ker
4      MicroFlex_e190_Ecat : ECAT_CiA402_Control_App;  (*Axis reference for MicroFlex_e190
5      MicroFlex_e190_axis : AC500_MotionControl.Axis_Ref := (
6          expert:=(
7              ECAT          := MicroFlex_e190_Ecat,
8              OperatingMode := 8,
9              Kernel        := MicroFlex_e190_Kernel
10             ),
11             parameter:=(
12                 paraReverseDirection      := 0,    //0 = normal direction, 1 = reverse input p
13                 paraEnablePosLagMonitoring := TRUE,    //Position lag super vision
14                 paraMaxVelocityAppl       := 36000,    //Maximum application velocity
15                 paraMaxAccelerationSystem  := 10000,    //Maximum acceleration
16                 paraMaxDecelerationSystem  := 10000,    //Maximum deceleration
17                 paraMaxJerk                := 2000    //Maximum Jerk
18             )); (*Axis class for MicroFlex_e190_axis*)
19      MicroFlex_e190_axis_ReadPos : AC500_MotionControl.MC_ReadActualPosition;
20      MicroFlex_e190_axis_ReadVel : AC500_MotionControl.MC_ReadActualVelocity;
21  END_VAR

```

The Parameters names in the generated code (Function block / Library) are different. Please refer the below table to find the name in wizard and respective name in the generated code (Function block / Library).

Name from Wizard	Name in generated code
Forward limit (Axis stop)	paraSWLimitPos
Reverse limit (Axis stop)	paraSWLimitNeg
Forward limit (Warning)	paraSWLimit2DecPos
Reverse limit (Warning)	paraSWLimit2DecNeg
Invert direction	paraReverseDirection
Position lag supervision	paraEnablePosLagMonitoring
Maximum application velocity	paraMaxVelocityAppl
Maximum acceleration	paraMaxAccelerationSystem
Maximum deceleration	paraMaxDecelerationSystem
Maximum jerk	paraMaxJerk
Following error percentage	Pos_Lag_percentage
Delay time velocity check	V_Check_Time
Control time	Control_Time
Feed forward percentage	FF_Percentage

Setting from EtherCAT master	Cycle = Cycle time set in the EtherCAT master in ms
Axis type	EN_Modulo = TRUE , when Modulo(Rotary) , Finite (Rotary) is selected.
Modulo Range (0-Value)	Modulo_Range = entered value from wizard is converted to encoder increment
Encoder increments per motor revolution	Inc_Per_R
Gearbox output turns	U_PER_REV_Nominator
Gearbox input turns	U_PER_REV_Denominator
Maximum speed reference value	Ref_MAX
Maximum speed	MAX_RPM




Note: It is recommended not to add/edit the program here manually since this will be lost next time when the user click on the "Generate Code" and if there was any modification done on the particular axis reference.

8.4.3.2 Generated Program (PRG)

Program which is generated by the wizard initiates the CMC_Axis_Control_Parameter, CMC_Basic_Kernel / CMC_Load_Motion_Kernel and ECAT_CiA402_Control_App function blocks and set the inputs based on the configuration done in the wizard.

For CSP and CSV control type, CMC_Basic_Kernel function block will be called and for CSVL control type CMC_Load_Motion_Kernel function block will be called.

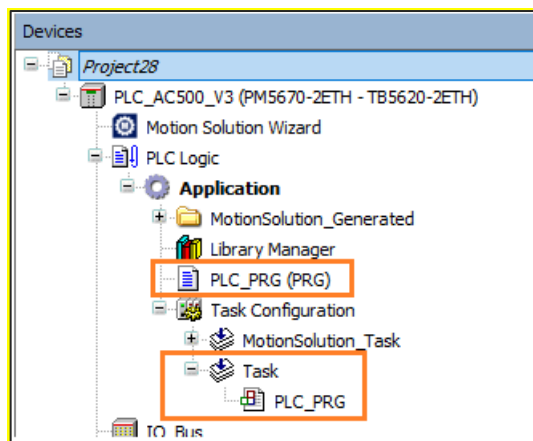


Note: It is recommended not to add/edit the program here manually since this will be lost next time when the user click on the "Generate Code" and if there was any modification done on the particular axis reference.

```

PRG_MicroFlex_e190_axis X
1  (* This POU has been auto generated by Automation Builder as part of Motion Solution wizard. *)
2  PROGRAM PRG_MicroFlex_e190_axis
3
4  VAR
5      MicroFlex_e190_Parameter : CMC_Axis_Control_Parameter := (
6          Enable := TRUE,
7          Pos_Lag_percentage :=150,
8          V_Check_Time :=T#100MS,
9          Control_Time :=100,
10         FF_Percentage :=50,
11         Cycle :=4,
12         EN_Modulo := TRUE,
13         Modulo_Range := 131072,
14         Inc_Per_R := 131072,
15         U_PER_REV_Nominator := 360,
16         U_PER_REV_Denominator := 1,
17         Ref_MAX :=13107200,
18         MAX_RPM :=6000 ); (*Axis reference for MicroFlex_e190_Parameter*)
19     MicroFlex_e190_Kernel : CMC_Basic_Kernel; (*Axis reference for MicroFlex_e190_Kernel*)
20     MicroFlex_e190_Ecat : ECAT_CiA402_Control_App; (*Axis reference for MicroFlex_e190_Ecat*)
21     MicroFlex_e190_Op_Mode : BYTE := 8; (*Axis reference for MicroFlex_e190_Op_Mode*)
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625

```



9 CAM EDITOR

The cam is integrated in the development interface of Automation Builder. In the cam editor, cams and tappets can be implemented graphically or by means of tables. As soon as code is generated for the corresponding application, global data structures ("Cam Data") are created which the IEC program can access.

9.1 Definition of a Cam

A cam describes the functional dependency of one drive (slave) on another drive (master). The relationship is described by a continuous function (or curve) that maps a defined range of master values to slave values.

You may also add tappets (binary switches) to the cam at any position. In this way, you can create cam tables which contain tappets.

9.2 Structure of the Cam Editor

Open the cam editor by double-clicking the "Cam" object in the device tree.

The editor consists of the following tabs:

- Tab "Cam" : Includes a graphical editor for creating a cam path. Here, you can display and modify the slave position, slave velocity, slave acceleration, and slave jerk. In the graphical editor, you recognize very quickly when you program a movement with high acceleration.
- Tab "Cam table" : Includes an editor for listing base points in a table. Here, you can specify the exact positions and velocities.
- Tab "Tappets" : Includes an editor for programming tappets (switch points) in a diagram. This display provides a very good overview of the sequential order of the tappets.
- Tab "Tappet table" : Includes an editor for listing switch points in a table. Here, you can specify the exact switch points.

The tabs are split into an editor, as well as a "ToolBox" view and "Properties" view.

9.2.1 Tab 'Cam'

In this graphical editor, the cam graphs are defined. You can switch between the graphical editor and the alternative tabular editor at any time ("Cam table tab" tab).



The editor window displays the curves of four graphs:

- Slave position (black)
- Slave velocity (blue)
- Slave acceleration (green)
- Slave jerk (yellow)

The horizontal axis of all four coordinate systems shows the range of the master values ([0,360]). The vertical axis in the position diagram shows the value range that is defined in the cam properties. The vertical axis of velocity, acceleration, and jerk is scaled automatically.

You can modify all curves, except the jerk curve. As velocity, acceleration, and jerk are derived curves, changes to one graph causes changes to the other graphs. You change the height of the diagram by moving the horizontal separation bars.

"View 'ToolBox'"

 "Select"	Select a line in the table by using this tool. Selected points are deleted by pressing the [Del] key.
 "Add point"	Add new points with this tool. Click the insertion point in the diagram. The graph is then adapted automatically so that its curve runs through the new inserted point.

“View 'Properties'”



X	X-position of the slave axis
Y	Y-position of the slave axis
V	Velocity of the slave axis
A	Acceleration of the slave axis
J	Jerk of the slave axis

9.2.2 Tab 'Cam table'


The cam table is an alternative to the graphical editor for defining the cam graphs (“Cam” tab).

You can switch between the table editor and the graphical editor at any time.

The first line of the table always contains the start position of the master (and the related slave values) and the last line is always the end position. The lines in-between alternately define segments and points.

	Inserts a new line.
	Deletes the selected segment
“X”	X-position of the slave axis
“Y”	Y-position of the slave axis
“V”	Velocity of the slave axis
“A”	Acceleration of the slave axis
“J”	Jerk of the slave axis
“Segment type”	<ul style="list-style-type: none"> • “Poly5”: 5th degree polynomial • “Line” • Linear
The following values result from the values of the respective segment. They cannot be modified.	
min(Position)	Minimum value of the slave position
max(Position)	Maximum value of the slave position
max(Velocity)	Maximum value of the velocity of the slave, based on the master axis
max(Acceleration)	Maximum value of the acceleration of the slave, based on the master axis


“View 'ToolBox'”

 “Select”	<p>Select a line in the table by using this tool.</p> <p>Selected points are deleted by pressing the [Del] key.</p>
--	---




9.2.3 Tab 'Tappets'

The tappet paths are defined in this table graphical editor. A tappet path defines one or more tappets depending on the master position. At the upper edge of the editor window, a horizontal axis approaches the range of the master positions. The individual tappet paths are defined below.

You can switch between the graphical editor and the alternative tabular editor at any time (“Tappet table” tab).

	<p>Track ID of the tappet path</p> <p>All tappets of a tappet path refer to the same tappet switch (a variable of type BOOL).</p>
---	---

















"View 'ToolBox'"

 : "Select"	<p>Select the tappets by means of this tool. You can drag the selected tappets to another position.</p> <p>You can modify the switch on/off attribute of a tappet by clicking the relevant end of the crossed line ().</p> <p>Delete the selected tappet by pressing the [Del] key.</p>
	Add new tappets with this tool. Click the insertion point in the path.

"View 'Properties'"

the tappet is assigned to a result, if it is passed from the position of the master axis in the positive (increasing master values) or negative direction.	
"X"	Position of the tappet
"Positive pass"	Switch on/off attribute <ul style="list-style-type: none"> • No action • Switch to ON • Switch to OFF • Invert
"Negative pass"	Switch on/off attribute <ul style="list-style-type: none"> • No action • Switch to ON • Switch to OFF • Invert



Table of the possible combinations of tappet attributes

Tappet symbol	Positive pass	Negative pass
	No action	No action
	Switch to ON	No action
	Switch to OFF	No action
	No action	Switch to ON
	No action	Switch to OFF
	Switch to ON	Switch to OFF
	Switch to ON	Switch to OFF
	Switch to OFF	Switch to ON
	Switch to OFF	Switch to OFF
	Invert	No action
	No action	Invert
	Switch to ON	Invert
	Invert	Switch to ON
	Invert	Switch to OFF
	Switch to OFF	Invert
	Invert	Invert

9.2.4 Tab 'Tappet table'

This tabular editor is an alternative to the graphical editor for configuring the tappet paths ("Tappets" tab). A tappet path defines one or more tappets depending on the master position. In the table, the lines with the definitions of the associated tappets follow below each line that defines a tappet path.

You can switch between the table editor and the graphical editor at any time.

	Inserts a new tappet.
	Deletes the tappet.
"Track ID"	ID of the tappet path All tappets of a tappet path refer to the same tappet switch (a variable of type BOOL).
"X"	Position of the tappet
"Positive pass"	Switch on/off attribute <ul style="list-style-type: none"> • No action • Switch to ON • Switch to OFF • Invert
"Negative pass"	Switch on/off attribute <ul style="list-style-type: none"> • No action • Switch to ON • Switch to OFF • Invert

"View 'Properties'"

The tappet is assigned to a result, if it is passed from the position of the master axis in the positive (increasing master values) or negative direction.	
"X"	Position of the tappet
"Positive pass"	Switch on/off attribute <ul style="list-style-type: none"> • No action • Switch to ON • Switch to OFF • Invert
"Negative pass"	Switch on/off attribute <ul style="list-style-type: none"> • No action • Switch to ON • Switch to OFF • Invert

9.2.5 Dialog 'Properties - 'Cam''

Use this dialog to define the global variables of the cam.

"Dimensions"

"Master start/end position"	The start and end positions of the master define the range of the master values and <u>therefore</u> the scale of the horizontal axis of the cam. The default settings are given in angular degrees with 0 and 360 as limiting values.
"Slave start/end position"	The associated slave positions are determined by the graph type that is defined for the cam. However, the segment depicted by the curves (this is also the scale of the vertical axis) can be defined by the start and end positions of the slave that are given here.

"Period"

These settings affect the work in the cam editor and cam table. Depending on these parameters, the slave start point is adjusted automatically when the end point is changed, as well as the other way around. This adjustment optimizes the period transition to be as smooth and jerk-free as possible.	
"Smooth transition"	<input checked="" type="checkbox"/> The values for position, velocity, and acceleration are adjusted automatically.
"Slave period"	Indicates when the slave period is repeated mechanically. Then the slave position at the start and end of the master period can deviate by one integer multiple of this value. This value is effective only if the "Smooth transition" check box is selected.

"Continuity requirements"

Activation of these options for the continuity of the curve does not have any effect when editing the cam. It does, however, prompt a continuity check, which reports any violations to the message view (CAM). It is not possible to edit jumps in the position curve. The default setting also requires the continuity of velocity and acceleration. You can clear these options, for example in the special case of a curve that consists of only linear segments. However, this can lead to breaks in the position curve. By default, the jerk (4th derivative) is not tested for jumps.	
"Position"	<input checked="" type="checkbox"/> The entire curve is tested for jumps.
"Velocity"	
"Acceleration"	
"Jerk"	

"Compile format"

When compiling, cam is described according to the following options:	
"Polynomial (XYVA)"	Polynomial description of the individual points, consisting of master position, slave position, slave velocity, and slave acceleration.

9.3 Creating Cams

The steps for creating a cam are explained by means of a sample application that describes a rotary table with eight slots (45° division). Inside, there is a component that is fused ultrasonically. The welding tool is fed in by a linear drive after the rotary table has turned. After welding, the linear axis returns, and the rotary table continues turning.

Work steps

- Rotary table turns 45° (duration: 400 ms).
- The welding head is moved down by a vertical axis of 250 mm (duration: 200 ms).
- Start welding (duration: 1200 ms).
- The welding head is moved up by a vertical axis of 250 mm (duration: 200 ms).

A cycle time of 2000 ms results from total times.

The application is implemented by means of a virtual master axis that runs continuously (modulo). The end value of the axis is projected according to the cycle time of 2000 ms. The rotary table is achieved as a cam (modulo; end value: 45°). The vertical axis is also achieved as a cam (restricted; end value: 300 mm). The welding process is controlled by a tappet.

9.3.1 Adding a cam to the device tree

Requirement: A AC500 controller is selected.

- Select the "Application" object in the device tree.
- Click "Project ▢ Add object ▢ Cam table".
- Specify the name "Rotary table" for the cam and click "OK".
 - The object is inserted into the device tree. The cam editor opens.
- Insert another cam named "Vertical axis".

9.3.2 Setting the properties of the cam

- Select the “Rotary table” cam in the device tree.
- Click “Properties” in the “View” menu or in the context menu.
- Select the “Cam” tab.
- Specify the following values:
 - “Master start position” : 0
 - “Master end position” : 2000
 - “Slave start position” : 0
 - “Slave end position” : 45
 - “Smooth transition” : (deactivated)
- Click “OK” to close the dialog. Confirm the dialog for changing the cam object. 6. Change the values for the “Vertical axis” cam in the same way:
 - “Master start position” : 0
 - “Master end position” : 2000
 - “Slave start position” : 0
 - “Slave end position” : 300
 - “Smooth transition” : (activated)
- Click “OK” to close the dialog. Confirm the dialog for changing the cam object.

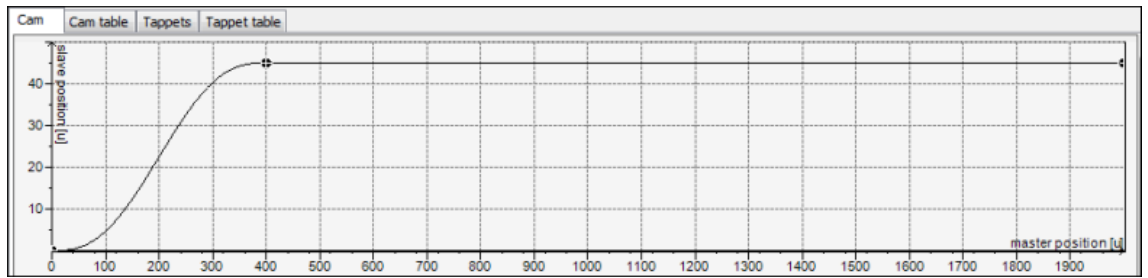
9.3.3 Changing the Cam Path

These instructions use the example from the section "Creating Cams" to demonstrate how to change a cam.


9.3.3.1 Changing the path with the graphical editor

1. Open the “Rotary table” cam in the editor.
 - The “Cam” tab is visible.
2. Select the point at 120 and delete it by pressing the delete key (*[Del]*). Also delete the
 - point at 240.
3. Select the “Add point” tool from the “ToolBox” view.
 - The mouse pointer turns into crosshairs when you move it into the editor.
4. Click near “Master position” 400 and “Slave position” 45 in the upper graphs (slave position).
5. The curve of the slave position is changed. The curves of velocity, acceleration, and jerk also change.
6. Select the new inserted point by clicking it.
7. Drag the point to another position.
 - The curve of the slave position is adjusted accordingly.
8. Change the “X” and “Y” properties to the exact values of 400 and 45, respectively.
9. In the same way, change the x-value to 45 of the point at master position 2000.
10. Select the “Select” tool from the “ToolBox” view.
11. Select the second curve element (between 400 and 2000). 11. Change the “Segment type” property to “Line”.
12. Check the curve in the graphical editor.

- Display:



9.3.3.2 Changing the path with a cam table

1. Open the “Vertical axis” cam in the editor.
 - The “Cam” tab is visible.
2. Select the “Cam table” tab.
3. Delete the point at 120 by clicking the  symbol. Also delete the point at 240.
4. Click the “+” symbol.
 - A new point and a new segment are inserted at (1000/150).
5. Add two more points.
6. Change the values X / Y of the following points:
 - Point 1: 0 / 0
 - Point 2: 400 / 0
 - Point 3: 600 / 250
 - Point 4: 1800 / 250
 - Point 5: 2000 / 0
 - The curve of the slave position is changed. The curves of velocity, acceleration, and jerk also change.
7. In the cam table, change the “Segment type” of the first and third segments to “Line”.
8. Check the curve in the graphical editor.



Note: By clicking “Display generated code”, you can display the automatically created global variables.

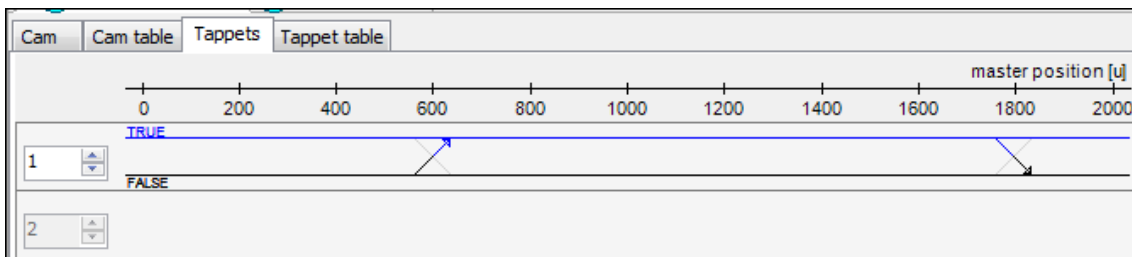
9.3.4 Defining Switch Points

Use switch points to trigger events depending on the master position. For example, this can be the setting of an output or the calling of a function block.

These instructions use the example from the section "Creating Cams" to demonstrate how to define the points. In this example, the tappet starts and stops the welding process.

1. Open the “Vertical axis” cam in the editor.
 - The “Cam” tab is visible.
2. Select the “Tappets” tab.
3. Select the “Add tappet” tool from the “ToolBox” view.

- The mouse pointer turns into crosshairs when you move it into the editor.
4. Click below the master position near position 600.
 - A tappet is inserted to the tappet path 1.
 5. Select the tappet.
 6. Change the values of the tappet in the “*Properties*” view.
 - “X”: 600
 - “*Positiver pass*”: “*Switch ON*”
 - “*Negative pass*”: “*No action*”
 7. Insert another tappet to tappet path 1 at X: 1800.
 - “X”: 1800
 - “*Positiver pass*”: “*Switch OFF*”
 - “*Negative pass*”: “*No action*”
 8. Check the result.



Note: You can also change the values for “Positive pass” and “Negative pass” by clicking the respective end of the crosshairs.



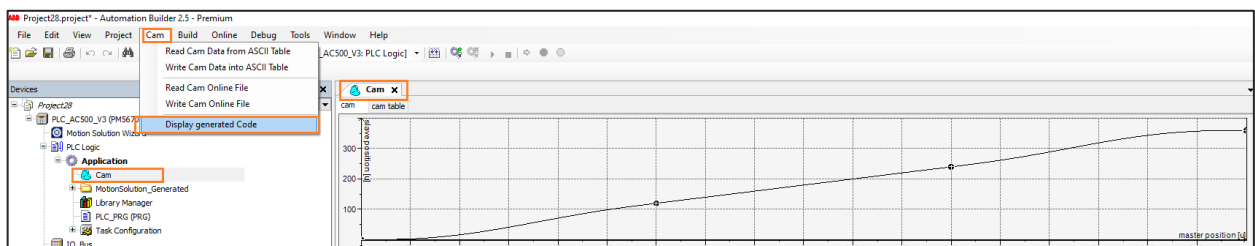
Note: Please note the possibility of also setting switch points in the “Tappet table” tab. This editor provides you with the same options, but in tabular form

Once the cam is added to the Automation Builder, user can double click on the same to visualize the cam in a graphical way by default or select the “cam table” tab from the same window if user wishes to see it in the table format.

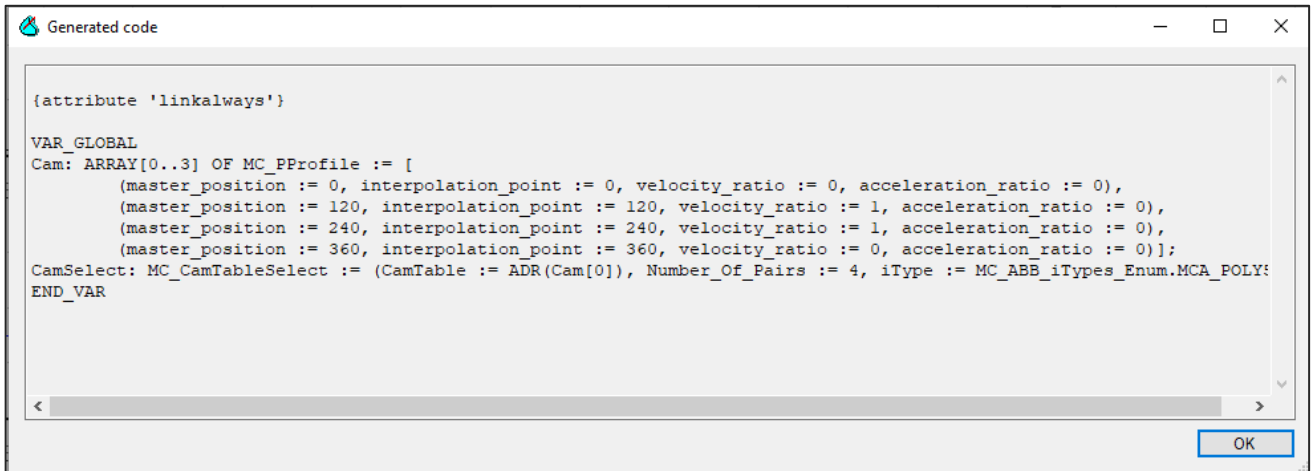
9.4 Cam generated code

If the user wants to see the generated code. There is an option to view the generated code which is created by the cam when the cam object is kept open. Currently it can only show the code generated from one cam which is open.

To view the code -> close all the cam views and open the cam which’s code needs to be viewed -> go to Cam tab from the main Automation Builder tab -> select “display generated code” option.



Code will be showed as below, where user can find the cam points are created in an array and cam table function block from motion library is already initialized with few inputs.



```

{attribute 'linkalways'}

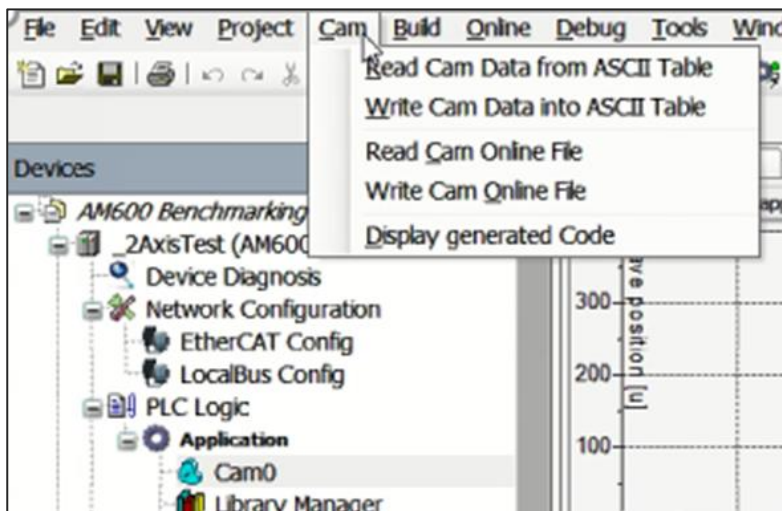
VAR_GLOBAL
Cam: ARRAY[0..3] OF MC_PProfile := [
    (master_position := 0, interpolation_point := 0, velocity_ratio := 0, acceleration_ratio := 0),
    (master_position := 120, interpolation_point := 120, velocity_ratio := 1, acceleration_ratio := 0),
    (master_position := 240, interpolation_point := 240, velocity_ratio := 1, acceleration_ratio := 0),
    (master_position := 360, interpolation_point := 360, velocity_ratio := 0, acceleration_ratio := 0)];
CamSelect: MC_CamTableSelect := (CamTable := ADR(Cam[0]), Number_Of_Pairs := 4, iType := MC_ABB_iTypes_Enum.MCA_POLY);
END_VAR
  
```

9.5 Importing a Cam from 3rd party Codesys controller

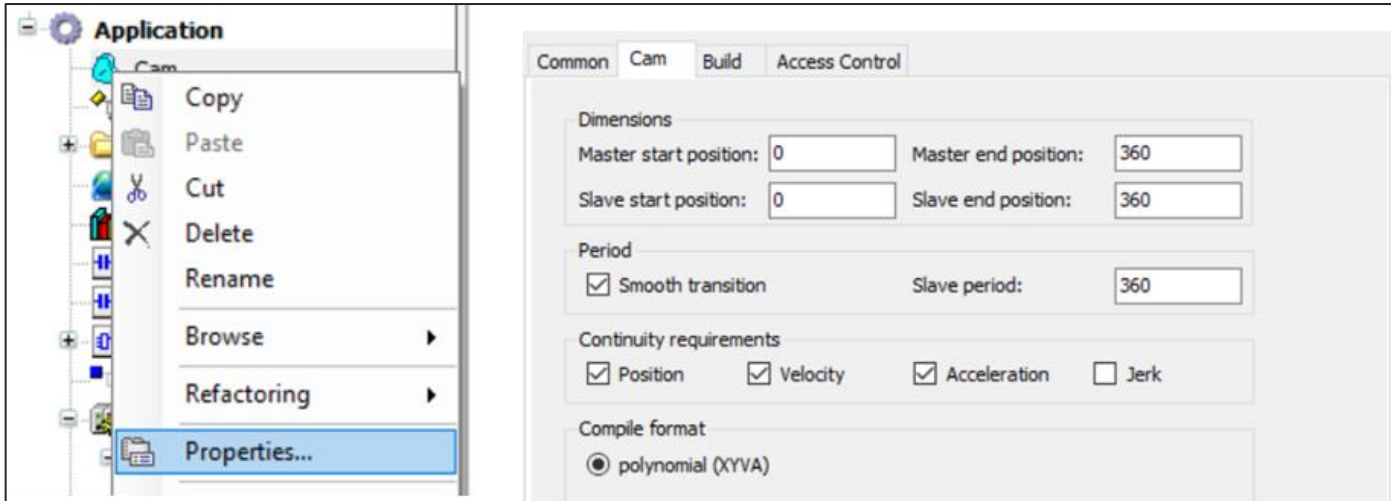
9.5.1 Exporting the Cam for the the 3rd party PLC

If you are using a Cam application that has been previously written using another 3rd party CoDeSys based PLC it's possible to reuse these same data points in your new Automation Builder project to save the user time and effort. The steps to do this are below, in the example application we are using a *Inovance AM600 PLC program*.

Open the 3rd party CoDeSys environment and the project in question. As when using ABB's solution, the Cam object must be selected from the project tree to allow the user to access the "Cam" item in the system menu at the top of the page which will be needed for some of the steps that follow.



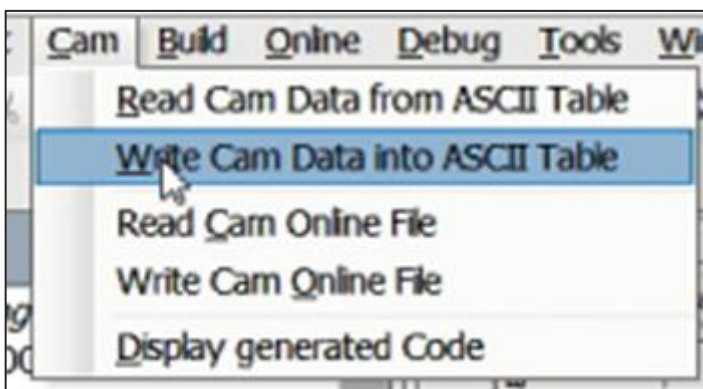
The first thing you should consider is the master and slave settings such as start and end position, and Slave period. To check this you can right click on the Cam object in the hardware tree and select properties. From here you will get a pop up where you can double check the settings that will define the cams operation. Make a note of these.



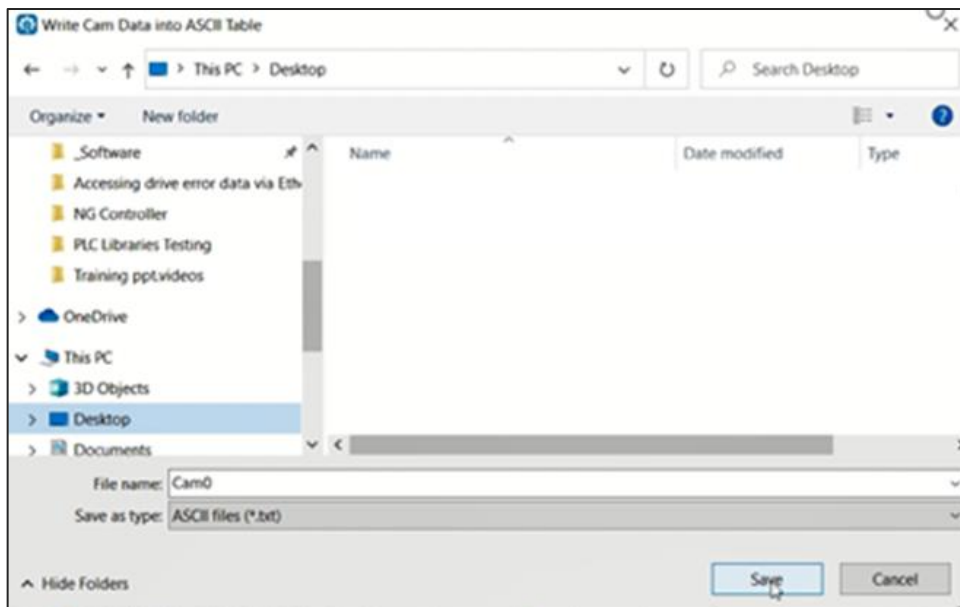
It's also a good idea to look at the Cam table just to get an idea about the General Layout, here we can see this Cam has 6 points (beginning, end and 4 user selected items)

[illegible]

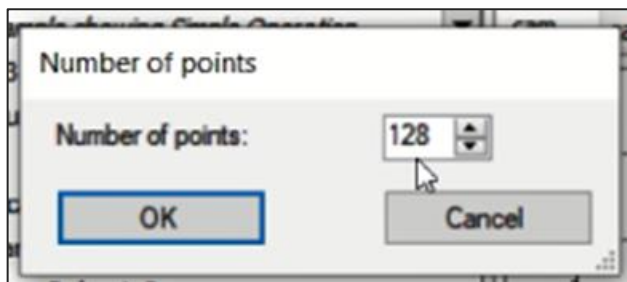
Next, we must consider the method for getting the Cam data exported from the application. To do this we must export or 'Write' the data to an ASCII file. To do this we must select the Cam object to access the Cam menu, then select the option for 'Write Cam Data into ASCII Table'



Then select somewhere to save it.



Then you will see a pop up which will ask you how many Cam data points you want to *export* into the file, we can treat this like resolution. In the example below we select 128 data points

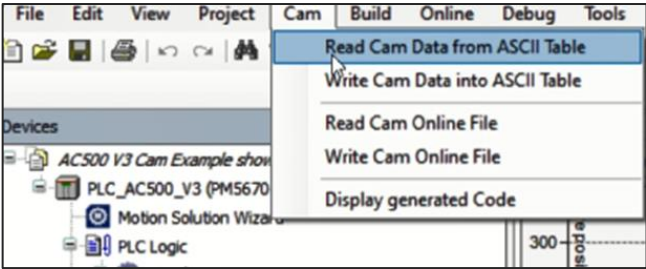


If we select many of points when the Cam is exported, we can expect the larger number of data points will give the same profile even using the 3rd party Cam builder's interpolation types to ensure the Cam profile is immediately matched perfectly to the previous application. If we select a lower number of points, one that matches the interpolation points (the smallest number possible) for example then you are relying on Automation builder to plot the interceding Cam points so the interpolation types selected within Automation Builder are more important and must be checked.

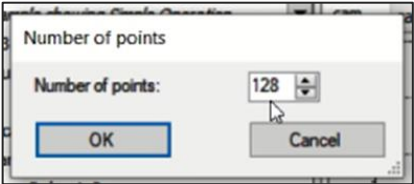
Now we are finished with the 3rd party environment.

9.5.2 Importing the Cam data into Automation Builder

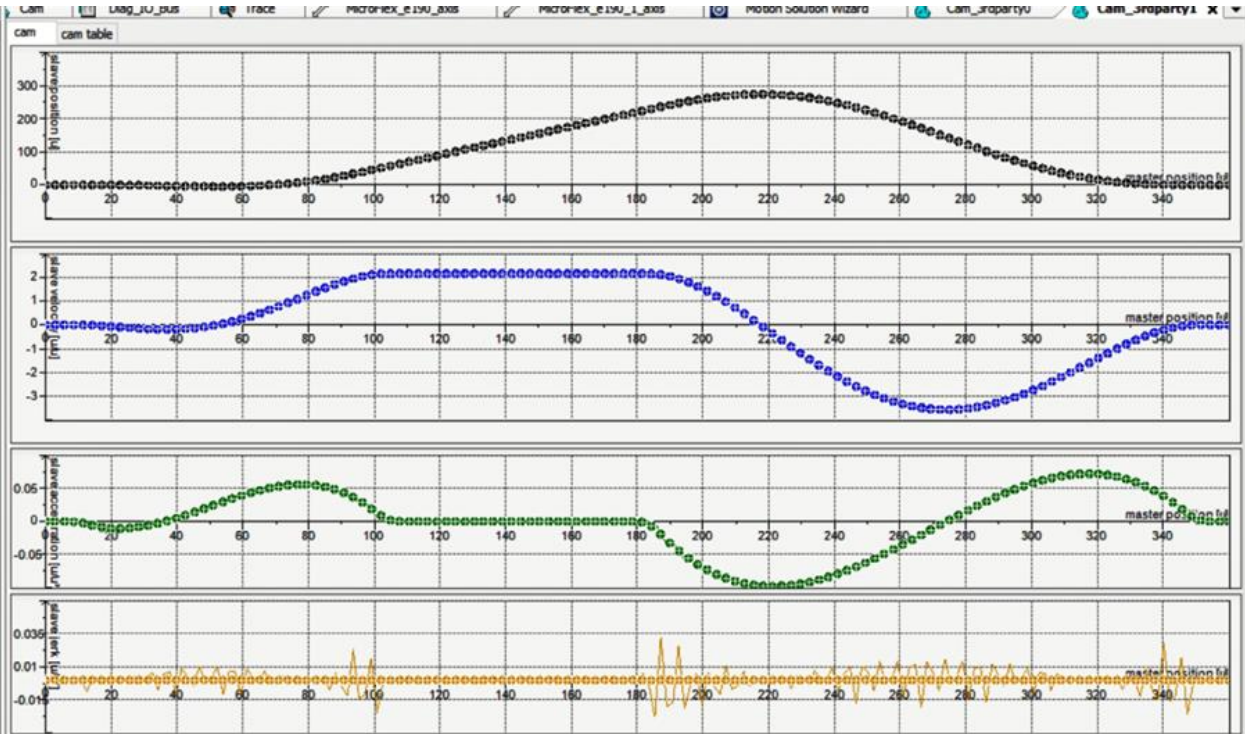
We can now switch to Automation builder in which we need to have a project which has a Cam in it ready for us to write the data into. As before we must check the properties of the Cam to check that it has the same maximum and minimum values and other Characteristics. Then we must select the Cam object to access the menu in the Tool bar from which we can select "Read Cam Data from ASCII table" and then select the ASCII file we created before.



You will then be asked how many data points you want to *Import* into the project. Typically, this should always match the value that was exported.



In the screen shot below you can see the result with all 128 data points imported



You can also edit the Segments (interpolation types) used in each segment in this newly imported Cam via the Cam table view.

	X	Y	V	A	J	Segme...	min(Po...	max(P...	max(V...	max(A...
	0	0	0	0	0					
	2.8346...	0	0	0	0	Poly5	0	0	0	0
	5.6692...	0	0	-2.3264...	0	Poly5	-3.2302...	0	4.4703...	2.3264...
	8.5039...	0	-0.0001...	-0.0006...	0	Poly5	-2.6924...	1.6136...	0.0001...	0.0006...

Once all necessary adjustments have been made, now the Cam is ready to be used in the program.

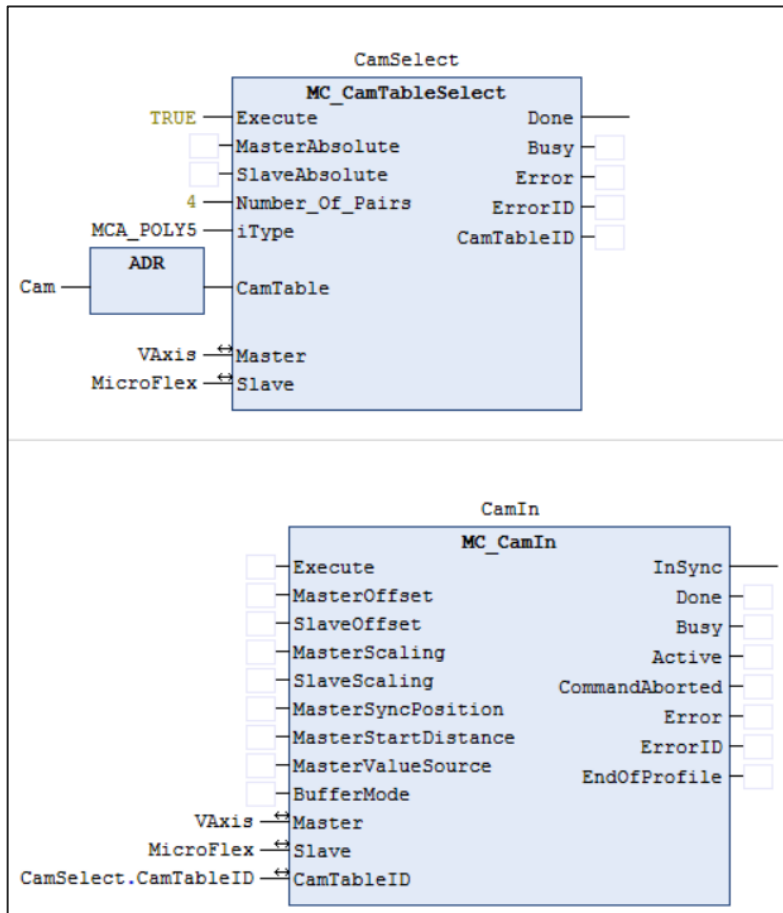
9.6 Application program using generated Cam

Once the cam is created by the cam editor, user can use the cam function blocks to use the same in user application. Users need to call at least the MC_CamTableSelect and a CamIn function block from the AC500_MotionControl library. An example program is shown below,

In below example we have used the MC_CamTableSelect instance generated by the cam editor itself, this helps the user not to set few of the input parameters since they are already set by the cam editor itself.

MC_CamTableSelect autogenerated instance name follow the Cam Editor object name + Select. For example, if the Cam editor object name is RotaryCam, MC_CamTableSelect instance name is “RotaryCam**Select**” and CamTable, Number_Of_Pairs and iType (MCA_POLY5) MC_CamTableSelect function block inputs are set by cam editor.

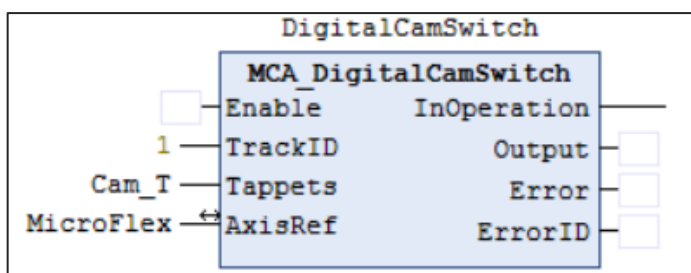
If the user created an different instance of MC_CamTableSelect than generated by Cam editor, user need to connect all the inputs as per the application requirement.



For the tappets, use the MCA_DigitalCamSwitch function block and configure the inputs.

The MCA_CAMTappet array is generated automatically based on the configuration in tappet. The array name follows the Cam object name with “_T”.

For example, if Cam object’s name is “RotaryShear”, MCA_CAMTappet array is generated as “RotaryShear_T”. This can be directly passed to input pin name “Tappets” of function block MCA_Tappet



More details on how to use the cam curve can be found in Automation Builder online help (PLC Automation with V3 CPUs > Libraries and solutions > Motion control library > PLC-based motion control > Basic functionalities > How to Use a CAM curve) and from the AC500_MotionControl library integrated documentation.

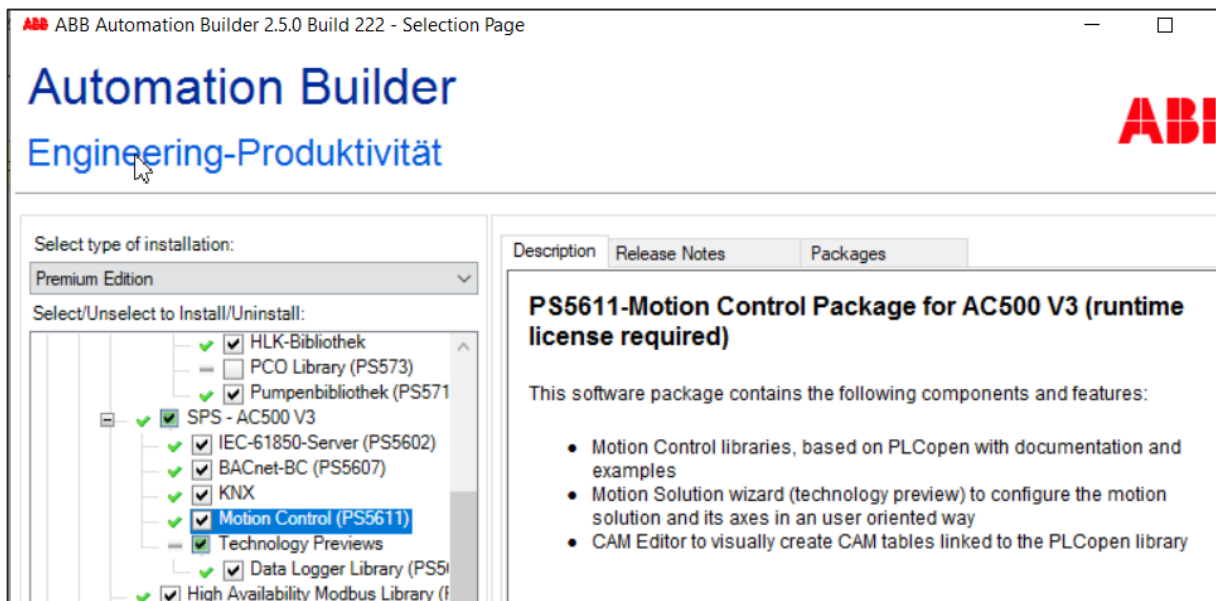
10 ABB PLCOPEN MOTION CONTROL LIBRARY

This chapter gives an overview of the libraries in the form of an “System Technology” in chapter (10.2), which explains the concepts used and lists the contents of the ABB PLCopen motion control libraries and the ABB specific additional libraries which are needed for motion control, interfacing, and e.g., special calculations in ABB PLCs.

The Function Blocks of the libraries themselves are documented then in chapters 10.3 for reference, the actual version of the libraries and their documentation can always be found within the libraries in Automation Builder (Library Manager).

The libraries are all included in Automation Builder via the optional install package “Motion Control (PS5611)”, which can be installed at any time via the Installation Manager. The core libraries need the runtime license PS5611-MC to be installed in the CPU.

The Motion Control package also contains the core functionality for the Motion Solution Wizard and Cam Editor described in a previous chapter.



The libraries function blocks descriptions are also visible directly in the library manager (access via Configuration tree). The library manager will always have the most up-to-date version matching the actual used library.

10.1 Motion Control library: System Technology

10.1.1 Preconditions for the use of the motion control libraries

The user has to read the following instructions and referenced documents before using the libraries:

The library package has been released for the software and firmware versions listed in the Readme file of Automation Builder (see “Help → Automation Builder Release Notes”). Please also check there for understanding updates and changes made in case you started with an older version.

In no event will ABB or its representatives be liable for loss of data, profits, revenue or consequential, incidental or other damage that may result from the use of other versions of product, software or firmware versions. The error-free operation of the motion control library with other devices, software or firmware versions should be possible but cannot be guaranteed and may need adaptations e.g. of example programs.

The Motion control package contains follows libraries

Library	Automation Builder	PLC Firmware
ABB_MotionControl_AC500	AB2.4.0 or higher	AC500 V3 firmware version 3.3.1 or higher / AC500-eCo V3 firmware version 3.4.0 or higher
ABB_Ecat_CiA402_AC500		
ABB_MathFunctions_AC500		
ABB_MotionControlEco_AC500 (Kernel blocks for Eco V3 PLCs)		
ABB_MotionControlLoad_AC500	AB2.5.0 or higher	AC500 V3 firmware version 3.5.0 or higher

10.1.2 Overview and Basics

The PS5611-MC is a Motion Control library for AC500 V3 CPUs, to create Motion Control applications based on Function Blocks according to the standard of PLCopen Motion Control. These function blocks can be used for PLC-based Motion Control and cover a wide range of possible Motion Control functionalities. Starting from single axis movements to master-follower axes to perform electronic gearing and CAM functions.

This documentation contains the following chapters:

In this Overview chapter information is provided for a better understanding of Motion Control with AC500 PLC and PS5611-MC libraries. There is also a tabular overview of the available PLCopen Function Blocks and their compatibility with PLC-based Motion Control and the provided drive-based Motion Control axis implementations.

The chapter PLCopen Introduction and Basics explains the principle of the PLCopen Motion Control standard as well as how PLCopen Function Blocks can be used to create PLC Motion Control application programs.

PLC-based Motion Control in AC500 is the next chapter, where it is explained how PLC-based Motion Control with AC500 can be realized and how it can be used in combination with the available PLCopen Function Blocks.

Finally the chapter Load Control/Torque Control: Fluid Power Extension according PLCopen explains how the PLCopen part 6 Fluid Power – extension, also called “load control”, can be used to practically realize a form of torque control (or -profiling) and how it can be used in combination with the available PLCopen Function Blocks to switch between torque/load control and position control.

10.1.2.1 PLC-based motion control

With PS5611-Motion the application program and the profile generator are realized in the PLC. The implementation of the profile generator is based on a set of Function Blocks which are named Central Motion Control (CMC as prefix).

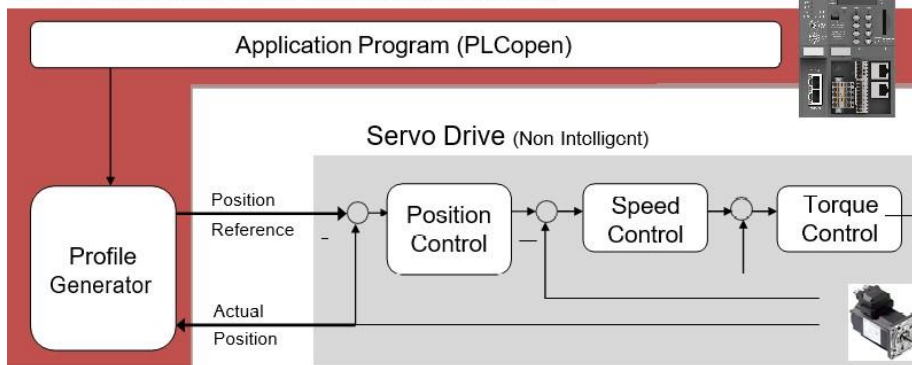
The profile generator of many possible axes is centrally placed inside the AC500 PLC. Therefore multi axis motion functionalities become easily available and can be accessed by PLCopen Function Blocks. As a result, Motion Control functionalities are almost drive independent.

Available motion control functionalities:

- Simple axis Movements
- Electronic Gearin
- Electronic CAMs
- Position Profiles
- Velocity Profiles
- Acceleration Profiles
- Load control (~ Torque profiling)

The output then is a position (or velocity) reference signal which the drive will follow. A new position or velocity reference value will be calculated with every cycle of the PLC and has to be transferred to the drive, which demands real time capabilities to the PLC and to the communication channel: A real time fieldbus like EtherCAT is needed. The feedback of the actual position can be used for supervision purposes during operation and is needed to adjust the value of the position reference before the drive will be enabled.

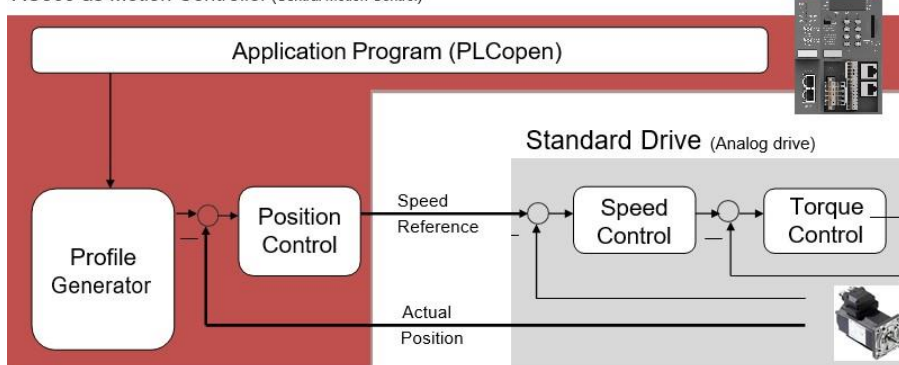
AC500 as Motion Controller (Central Motion Control)



System structure of PLC-based Motion Control with AC500 PLC and PLC-based motion control

With PLC-based Motion Control it is also possible to include the position control loop to the AC500 PLC. In this case a speed reference signal will be transferred to the drive, which makes it possible to perform the full range of motion functionalities with standard drives. To close the position control loop, feedback of the actual position is mandatory.

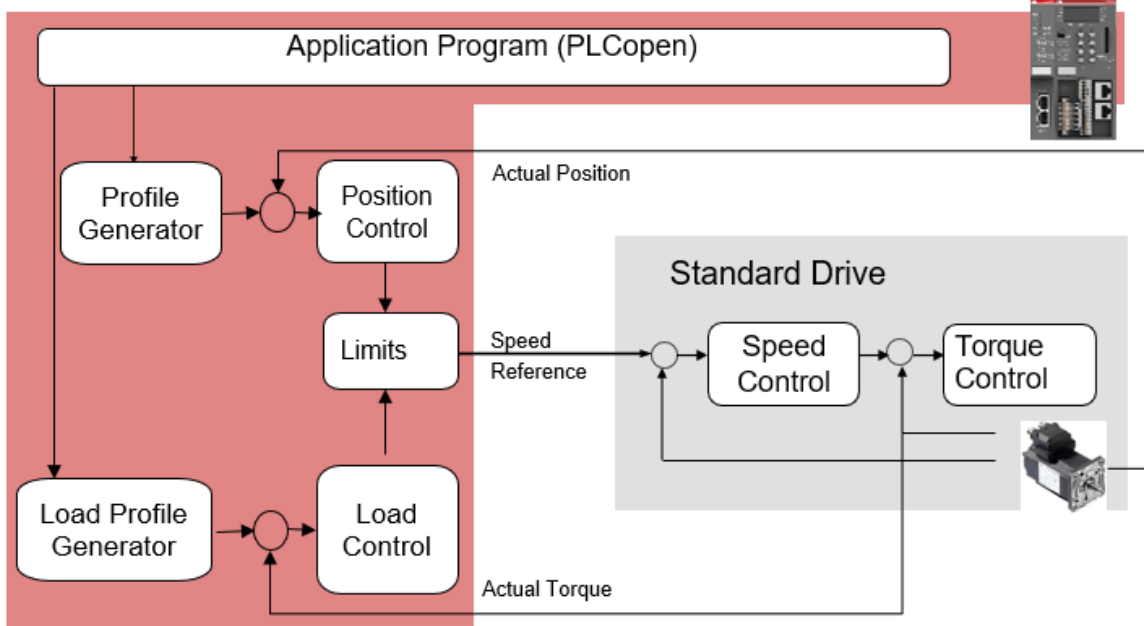
AC500 as Motion Controller (Central Motion Control)



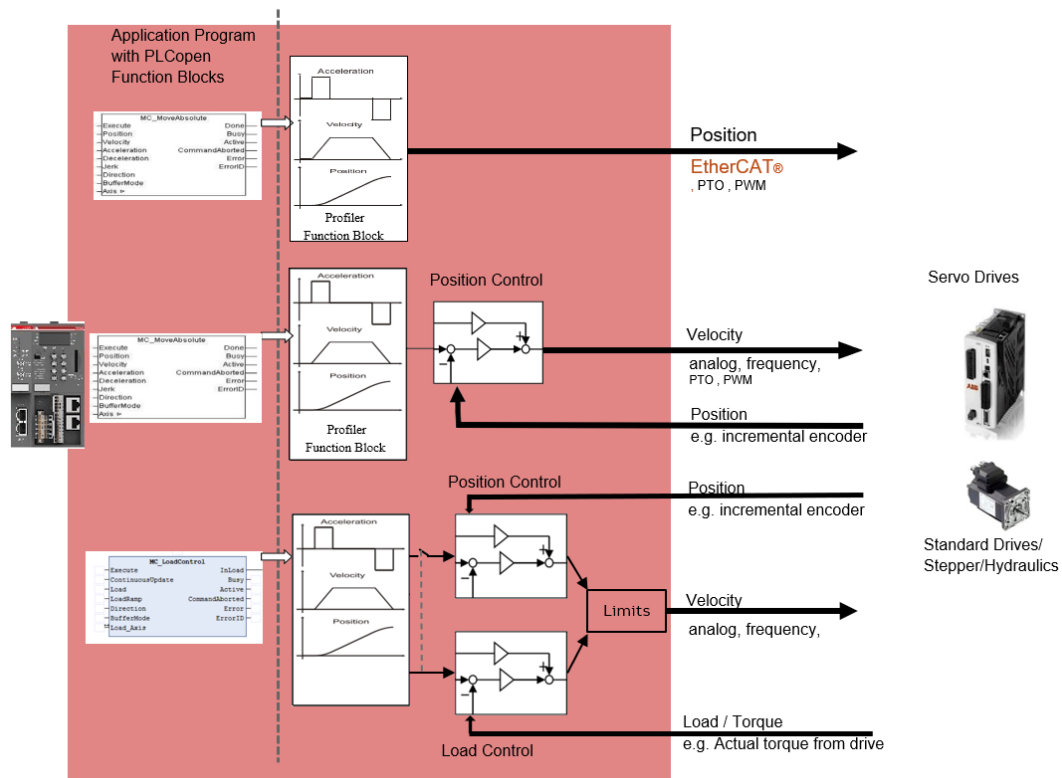
PLC-based Motion Control with AC500 PLC and PS5611-Motion, closed position control loop

With PLC-based Motion Control it is also possible to include the load control loop to the AC500 PLC. In this case a speed reference signal will be transferred to the drive, which makes it possible to perform the full range of motion functionalities with standard drives. To close the position control loop, feedback of the actual position is mandatory and to close the load control loop, feedback of the actual load / torque is mandatory.

AC500 as Motion Controller



PLC-based Motion Control with AC500 PLC and PS5611-Motion, closed load controlloop



Central Motion Control with AC500 PLC and PS5611-Motion, can realize different axis implementations at the same time

10.1.2.2 Overview of libraries

The following libraries are included in the PS5611-Motion package according to the listed use cases.

Library in Motion Control package	Description
ABB_MotionControl_AC500.compiled-library	Motion Control Library with PLCopen type function blocks: According PLCopen or ABB specific
ABB_MotionControlLoad_AC500.compiled-library	Function blocks for load / torque / pressure control functionality based on "PLCopen Motion Part 6 – Fluid Power Extensions"
ABB_Ecat_CiA402_AC500.library	CiA 402 EtherCAT state machine function block and additional EtherCAT function blocks adapted for Motion Control purposes (adaptable for 3rd party devices -> open/not protected)
ABB_MotionControlEco_AC500.compiled-library	Kernel function block to use PWM and PTO outputs from eCo V3 to realize PLCopen based motion.
ABB_MathFunctions_AC500.compiled-library	Mathematical functionality used for Motion function blocks

The features of the Function Blocks provided with PS5611-Motion can be used from the PLC program according to PLCopen standard. Different drives and different Motion Control realizations could be used and can be combined with each other as well as different fieldbuses.

ABB_Ecat_CiA402_AC500.library is editable and can be adapted based on the drive configuration and drive type.

10.1.2.3 Overview of PLCopen function blocks

The following tables give an overview of the available Function Blocks by library and category:

AC500_MotionControl	ABB Specific	MCA_CamGetInterpolationPosition	gives an interpolation result, acc. referenced cam table, for a master position.
		MCA_CamInDirect	implements Camming-Functionality. A device is coupled to a master
		MCA_CamInfo	gives an information which index is actually processed by the respective cam table
		MCA_Cam_Extra (Function)	to modify 2 mode-bits which define the behavior for the MC_CamIn more precise
		MCA_DriveBasedHome	to execute a homing procedure directly in the drive
		MCA_GearInDirect	commands a gear ratio btw. position of device and master from synchronization onwards
		MCA_Indexing	upon a rising edge, do a number of relative or absolute moves, listed in a table
		MCA_JogAxis	jogs an axis for a given distance fwd./backward with selected jog vel./acc.
		MCA_MoveByExternalReference	gives a reference position to axis which is directly passed to position control loop
		MCA_MoveRelativeOpti	commands a controlled motion of a distance relative to actual position
		MCA_MoveVelocityContinuous	commands a never-ending controlled motion at a specified velocity.
		MCA_Parameter	to change the default values of parameters
		MCA_PhasingByMaster	movement for the relation to the master axis of the specified axis (in sync)
		MCA_ReadParameterList	reads a list of parameters by using the "MC_ReadParameter".
		MCA_SetOperatingMode	changes the axis mode from positioning to velocity mode and vice versa.
		MCA_SetPositionContinuous	modifies the position of an axis with a defined profile
		MCA_WriteParameterList	writes a list of parameters by using the "MC_WriteParameter"
		MCA_DigitalCamSwitch	The output pin is Switched on- and off-based on TrackID, Axis Position and Configuration in the MCA_CAMTappet array
	MC Administrative	MCA_MoveBuffer	do a number of relative or absolute moves, listed in a table (Array of MCA_Pos_Ref).
		MCA_MoveByExtRefRelative	gives a reference position to the axis which is directly passed to the position control loop
		MC_CamTableSelect	selects the CAM tables by setting the connections to the relevant tables
		MC_Power	controls the power stage (on or off).
		MC_ReadActualPosition	returns the actual position
		MC_ReadActualVelocity	returns the value of the actual velocity as long as Enable is set.
		MC_ReadAxisError	describes general axis errors not relating to the PLCopen function blocks
		MC_ReadBoolParameter	transfers parameters defined as "standard" in the PLCopen
		MC_ReadParameter	transfers parameters defined as "standard" in the PLCopen
		MC_ReadStatus	returns detailed status of axis w. respect to motion currently in progress.
		MC_Reset	transition ErrorStop to StandStill by resetting all int. axis-errors
		MC_SetOverride	set a factor: multiplied to commanded vel., acceleration, deceleration., jerk of move FB.
		MC_SetPosition	shifts coordinate system by manipulating set-point, position, actual position with same value w/o movement.
		MC_WriteBoolParameter	transfers parameters defined as "standard" in the PLCopen
		MC_WriteParameter	transfers parameters defined as "standard" in the PLCopen
	MC Homing	MC_StepAbsSwitch	performs a homing by searching for absolute external physical switch
		MC_StepDirect	static homing by directly forcing an actual position. No physical motion performed
		MC_StepLimitSwitch	homing function by searching for sensor using only limit switches.
		MC_StepRefPulse	homing by searching for Zero pulse
	MC MultiAxis	MC_CamIn	implements Camming: A device- is coupled to a master axis by pos./pos. relation
		MC_CamOut	disengages the Slave axis from the Master axis immediately
		MC_CombineAxes	combines the motion of 2 axes into a third axis with selectable combination method.
		MC_GearIn	commands a ratio between the velocity of the slave and master axis.
		MC_GearInPos	commands a gear ratio btw. position of device, master axes from sync. point onw.
		MC_GearOut	disengages the Slave axis from the Master axis
		MC_HaltPhasing	commands a controlled motion stop for the phasing movement.
		MC_PhasingAbsolute	movement for the relation to the master axis of the specified axis.
		MC_PhasingRelative	movement for the relation to the master axis of the specified axis.
	MC SingleAxis	MC_AccelerationProfile	commands a time-acceleration locked motion profile.
		MC_Halt	commands a controlled motion stop.
		MC_HaltSuperImposed	commands halt to all superimposed motions. The underlying motion is not interrupted.
		MC_MoveAbsolute	commands a controlled motion to a specified absolute position.
		MC_MoveAdditive	commands a controlled motion of a specified relative distance additional to the most recent commanded position in the discrete motion state.
		MC_MoveContinuousAbsolute	commands controlled motion to specified abs. position ending with spec. velocity.
		MC_MoveContinuousRelative	commands controlled motion of specified rel. distance, ending with spec. velocity.
		MC_MoveRelative	commands controlled motion of spec. distance rel. to actual position at time of execution.

CMC_Blocks	MC_MoveSuperImposed	commands controlled motion of spec. Rel. distance additional to existing motion. The existing Motion is not interrupted but is superimposed.
	MC_MoveVelocity	commands a never-ending controlled motion at a specified velocity.
	MC_PositionProfile	commands a time-position locked motion profile.
	MC_Stop	commands a controlled motion stop and transfers the axis to the state "Stopping".
	MC_VelocityProfile	commands a time-velocity locked motion profile.
	CMC_Axis_Control_Parameter	provide basic information regarding underlying axis behavior and to configure the closed loop control for CMC_Basic_Kernel.
	CMC_Axis_Simu	can be used with CMC_Basic_Kernel to create a virtual axis.
	CMC_Basic_Kernel	Kernel is the fundamental FB of Central Motion Control axis implementation (floating point)
	CMC_Binary2Modulo	Convert a 32 bit value (Position_Reference) to Modulo_Range.
	CMC_Get_Units_From_Inc (Function)	converts drive's position value (DINT) exchanged with drive to scaled position unit (LREAL) used by the PLCopen FBs.
	CMC_Modulo2Binary	Convert a <32 bit value to 32 bit for use as Actual_Position.
	CMC_PidT1	PIDT1 Controller
	CMC_SIPosiLoop	simple interpolation. Alternatively, CMC_Sinterpolation can be used
	CMC_SinterPolation	used for a simple point-to-point interpolation, can be combined with CMC_SIPosiLoop as position control loop.

AC500_MotionControlLoad	CMC Block	MC_LimitLoad	activates a limitation of the load values provided by an axis.
		MC_LimitMotion	limits the motion values for the movement of an axis.
		MC_LoadControl	commands a controlled torque/force/pressure movement.
		MC_LoadProfile	commands a time-load locked motion profile. Load in the final element of the profile should be maintained
		MC_LoadSuperimposed	commands a controlled torque/force/pressure movement.
		MC_TorqueControl	commands a controlled torque/force/pressure movement and limits movement of an axis.(Wrapper FB)
AC500_MotionControlEco	eCo Kernel Function blocks	CMC_Load_Motion_Kernel	uses the CMC_Basic_Kernel for motion control regarding position and velocity, extends functionality to include load control
		OBIO_PTOMotionKernel	extends basic motion kernel functionality, to be used for eCo PTO outputs to connect a stepper drive
AC500_MotionControlEco	eCo Kernel Function blocks	OBIO_PWMotionKernel	extends basic motion kernel functionality, to be used for eCo PWM outputs to connect a stepper drive.
AC500_Math Functions		MATH_LINEAR_REGRESSION	calculates estimated next value based on a linear regression with 8 values history
AC500_Ecat_CiA402	CoE	ECAT_Read_Byte_App	reads an 8 bit value from an EtherCAT node. It uses EcatCoeRead to do so
		ECAT_Read_Coe_List_App	reads a list of parameters to the drive by using the EcatCoeRead.
		ECAT_Read_DInt_App	reads a 32 bit value from an EtherCAT node. It uses EcatCoeRead to do so
		ECAT_Read_Int_App	reads a 16 bit value from an EtherCAT node. It uses EcatCoeRead to do so
		ECAT_Write_Byte_App	writes an 8 bit value to an EtherCAT node. It uses EcatCoeWrite to do so
		ECAT_Write_Coe_List_App	writes a list of parameters to the drive by using the EcatCoeWrite.
		ECAT_Write_DInt_App	writes a 32 bit value to an EtherCAT node. It uses EcatCoeWrite to do so
		ECAT_Write_Int_App	writes a 16 bit value to an EtherCAT node. It uses EcatCoeWrite to do so
	Drive	ECAT_CiA402_Control_App	controls the state machine for a drive with 402 profile and connected to EtherCAT.
	Touch-Probe	ECAT_402Parameter-Homing_APP	sends parameters needed for homing to drive: all parameter with a value <> 0
		ECAT_HomingOnTouch-Probe_APP	performs homing on latched position value, e.g., Touch Probe on Z-pulse of encoder
		ECAT_CiA402_Touch-Probe_App	manages Touch Probe objects accord.: "EtherCAT Implementation Directive for CiA402

10.1.2.4 Overview of data types

The following data types are used for the Motion Control library. The data types are defined in the library file ABB_MotionControl_AC500 compiled-library. The corresponding elements can be used for the Function Blocks inputs.

Structures

Data type	Elements	Element data type
CMC_AXIS_IO	limitSwitchPos	BOOL
	limitSwitchNeg	BOOL
	absRefSwitch	BOOL
MC_PPROFILE Chapter "Position- PositionProfile"	master_position	LREAL
	interpolation_point	LREAL
	velocity_ratio	LREAL
	acceleration_ratio	LREAL
MC_TPROFILE Chapter "PositionTimeProfile"	interpolation_point	LREAL
	first_derivative	LREAL
	second_derivative	LREAL
	delta_time	TIME

Enumeration

Data type	Possible values
MC_ABB_iTYPES_ENUM Chapter "Interpolation types for profiles"	MCA_SPLINE_COMPLETE
	MCA_SPLINE_NATURAL
	MCA_POLY5
	MCA_POLY3
	MCA_LINEAR
MC_BUFFERMODE	mcABORTING
	mcBUFFERED
	mcBLENDINGlow
	mcBLENDINGprevious
	mcBLENDINGnext
	mcBLENDINGhigh
MC_DIRECTION	DEFAULT
	POSITIVE
	SHORTEST
	NEGATIVE
	CURRENT
	POSITIVE_STOP
	NEGATIVE_STOP
	CURRENT_STOP
MC_HOMING_DIRECTION	MC_SwitchNegative
	MC_SwitchPositive
	MC_Positive
	MC_Negative
MC_HOMING_EDGE	MC_EdgeOn
	MC_EdgeOff
	MC_On
	MC_Off
MC_HOMING_MODE	MC_REFPULSE
	MC_DIRECT
MC_SOURCE	mcActualValue
	mcSetValue
ERROR_ID	MC_Ok
	Wrong_State
	Drive_Problem
	Parameter_Exceeds_Limit
	No_Field_Access
	Bus_Problem
	Abs_Switch_Error
	Timeout
	NAK
	MC_TimeLimitExceeded
	MC_DistanceLimitExceeded
	MC_TorqueLimitExceeded
	Not_Implemented
	ErrorID_POSITION_FOLLOW
	ErrorID_POSSW
	ErrorID_NEGSW
	ErrorID_VELOCITY_FAULT
	ErrorID_INTERPOLATION_FAULT
	ErrorID_WARNING_VELOCITYLIMIT
	ErrorID_WARNING_POSITIONLIMITPOS
	ErrorID_WARNING_POSITIONLIMITNEG
	ErrorID_WARNING_POSITIONOVERRUN
	ErrorID_WARNING_ABORT
	ErrorID_WARNING_MOVEMENT_DIRECTION

10.1.2.5 Naming of function blocks and data structures

All Function Blocks and data types named MC_xxx are implemented according to PLCopen definition and follow the PLCopen documentation. They may have additional inputs but according to PLCopen rules.

All Function Blocks and data types named MCA_xxx are implemented corresponding to PLCopen rules with adaptations specific to AC500. They are AC500 specific extensions to the PLCopen library.

All Function Blocks named CMC_xxx belong to the implementation of PLC-based Motion Control.

All data types named CMC_xxx belong to the implementation of PLC-based Motion Control.

All data types named AXIS_xxx exist according to PLCopen definition. The content is ABB specific and not documented.

All Function Blocks named zCMC_xxx belong to the implementation of PLC-based Motion Control. These are not documented and not intended for customer use.

All function blocks and data types named OBIO_xxx in the ABB_MotionControlEco_AC500 library are intended for use with AC500-eCo V3 PLCs only.

All Function Blocks named xxx_APP are not write-protected and may be modified for adaptations. Then an editable library is available in the Example folder.

10.1.3 PLCopen Introduction and Basics

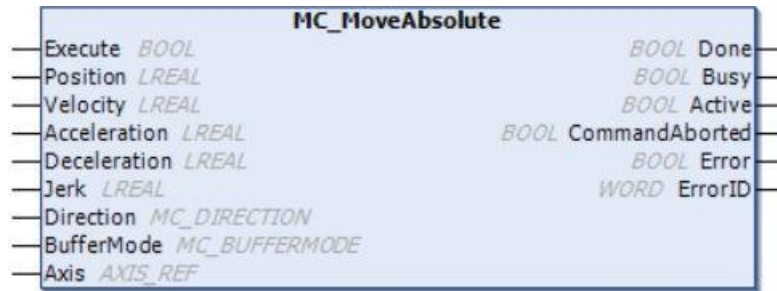
Based on application requirements and project specifications engineers are required to use or select a wide range of Motion Control hardware. In the past this required unique software to be created for each application even though the functions are the same. PLCopen motion standard provides a way to have standard application libraries that are reusable for multiple hardware platforms. This lowers development, maintenance and support costs while eliminating confusion. In addition, engineering becomes easier, training costs decrease, and the software is reusable across platforms. Effectively, this standardization is done by defining libraries of reusable components. In this way the programming is less hardware dependent, the reusability of the application software is increased, the cost involved in training and support reduced, and the application becomes scalable across different control solutions. Due to the data hiding and encapsulation, it is usable on different architectures, for instance ranging from centralized to distributed or integrated to networked control. It is not specifically designed for one application but will serve as a basic layer for ongoing definitions in different areas. As such it is open to existing and future technologies.

ABB is a member of the PLCopen organization. More Information about PLCopen can be read on the [PLCopen website](#).

Function Blocks according to PLCopen are designed for controlling axes via the language elements consistent with those defined in the IEC 61131-3 standard. It was decided by the task force that it would not be practical to encapsulate all the aspects of one axis into only one Function Block. The retained solution is to provide a set of command-oriented Function Blocks that have a reference to the axis, e.g. the abstract data type Axis, which offers flexibility, ease of use and reusability.

Implementations based on IEC 61131-3 (for instance via Function Blocks and SFC) will be focused towards the interface (look-and-feel/proxy) of the Function Blocks. This specification does not define the internal operation of the Function Blocks.

PLCopen Motion Control Function Blocks can be used in any IEC 61131-3 programming language. The following picture shows an example of a Function Block used in Function Block Diagram (FBD) language.



Command for absolute positioning according to PLCopen standard

Application programs which use the manufacturer independent function blocks according toPLCopen will lead to the following advantages:

- Reusable software structure for different platforms.
- Programming based on Function Blocks.
- Function Blocks can be used in any IEC 61131-3 language.

All function blocks which are defined by PLCopen will have the following qualities independently to the manufacturer of the motion control system:

- Same inputs/outputs
- Same functional behavior
- Same name

The following parts of the PLCopen motion control definition are completely or partly included in this product:

- Part 1: Function Blocks for Motion Control
- Part 2: Extensions
- Part 3: User Guidelines (partly)
- Part 4: Homing Procedures (partly)
- Part 6: Function Blocks for Motion Control – Fluid Power Extensions

10.1.3.1 Programming guidelines

This chapter explains some rules on the usage of libraries and the structure Axis_Ref.

In general, the kernel function block and the transfer of axis IO data should be processed in a cyclic task. This task should be as short and real-time as possible to achieve the best motion control performance. Always make sure Kernel function block is called at the highest priority task and other applications must be at a lower priority task.

If Axis_Ref is used as input on a user defined Function Block or program or function, always use it as VAR_IN_OUT and never use it as VAR_INPUT or VAR_OUTPUT. The reason is that this would

- Break the consistency and destroy data.
- Consume a lot of computing power by copying data.

Any instance of a Function Block should be called only once per cycle and in only one specific task.

If the instance is used in several tasks, it must be checked that it is not called several times. Because this could corrupt the handshake from Function Block to Axis_Ref to CMC_Basic_Kernel and vice versa.

Some PLCopen Function Blocks are only allowed to be called within the same task as theCMC_Basic_Ker-
nel Function Block. This is mentioned in the Function Block descriptions.

If PLCopen Function Blocks are called from a different task, the cycle time should be atleast 2 times the cycle time for CMC_Basic_Kernel Function Block.

10.1.3.1.1 Axis data type Axis_Ref

The Axis_Ref is a structure that contains information on the corresponding axis. It is used as a VAR_IN_OUT in all Motion Control Function Blocks defined in this document. The content of this structure is implementation dependent and can be empty. If there are elements in this structure, access to them is supported, but further details are outside of the scope of this document. The refresh rate of this structure is also implementation dependent. According to IEC 61131-3 it is allowed to switch the Axis_Ref for an active Function Block, for instance from Axis1 to Axis2. However, the behavior of this can vary across different platforms, and is not encouraged to do.

Axis_Ref data type declaration:

TYPE Axis_Ref : STRUCT

(Content is implementation dependent)

END_STRUCT

Example:

TYPE Axis_Ref : STRUCT

AxisNo: UINT; AxisName: STRING (255);

.....

END_STRUCT

10.1.3.2 The single axis state diagram

The diagram defines the behavior of the axis at a high level when multiple motion control Function Blocks are simultaneously activated. This combination of motion profiles is useful in building a more complicated profile or to handle exceptions within a program. (In real implementations there may be additional states at a lower level defined). The basic rule is that motion commands are always taken sequentially, even if the PLC had the capability of real parallel processing. These commands act on the axis' state diagram.

The axis is always in one of the defined states (see diagram below). Any motion command that causes a transition changes the state of the axis and, therefore, modifies the way the current motion is computed. The single axis state diagram is an abstraction layer of what the real state of the axis is, comparable to the image of the I/O points within a cyclic (PLC) program. A change of state is reflected immediately when issuing the corresponding motion command.

The diagram is focused on a single axis. The multiple axis Function Blocks, MC_CamIn, MC_GearIn and MC_Phasing, can be looked at, from a single axis state diagram point of view, as multiple single axes all in specific states. For instance, the CAM-master can be in the state Continuous Motion. The corresponding slave is in the state Synchronized Motion. Connecting a slave axis to a master axis has no influence on the master axis.

The state Disabled describes the initial state of the axis. In this state the movement of the axis is not influenced by the Function Blocks. The axis feedback is operational. If the MC_Power Function Block is called with Enable=TRUE while being in state Disabled, this either leads to Standstill if there is no error inside the axis, or to ErrorStop if an error exists.

Calling MC_Power with Enable=FALSE in any state, the axis goes to the state Disabled, either directly or via any other state. If a motion generating Function Block controls an axis, while the MC_Power Function Block with Enable=FALSE is called, the motion generating Function Block is aborted (CommandAborted).

The intention of the state ErrorStop is that the axis goes to a stop, if possible. There are no further inputs from Function Blocks accepted until a reset has been done from the ErrorStop state.

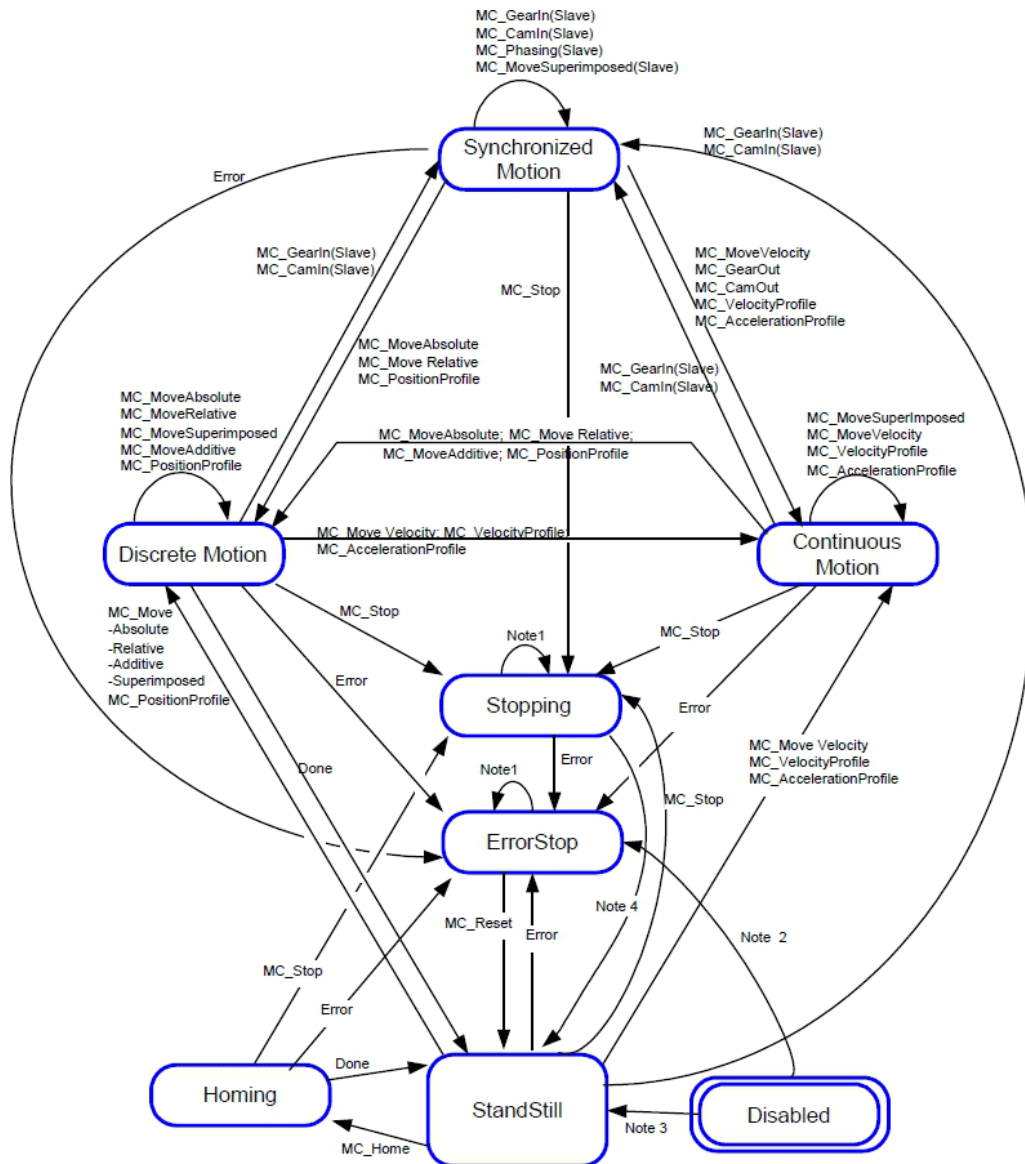
The transition Error refers to errors from the axis and axis control, and not from the Function Block instances. These axis errors may also be reflected in the output of the Function Blocks instances errors.

Issuing MC_Home in any other state than StandStill will go to ErrorStop, even if MC_Home is issued from the state Homing itself.

Function Blocks which are not listed in the single axis state diagram do not affect the state of the axis, meaning that whenever they are called the state does not change. They are:

MC_ReadStatus; MC_ReadAxisError; MC_ReadParameter; MC_ReadBoolParameter; MC_WriteParameter; MC_WriteBoolParameter; MC_ReadActualPosition and MC_CamTableSelect.

Calling the Function Block MC_Stop in state StandStill changes the state to Stopping and back to Standstill when Execute = FALSE. The state Stopping is kept if the input Execute is TRUE. The output Done is set when the stop ramp is finished.



Function Block state behavior



Note:

1. In this state ErrorStop or Stopping, all Function Blocks can be called, although they will not be executed, except MC_Reset and Error – they will generate the transition to StandStill or ErrorStop respectively.
2. Power.Enable=TRUE and there is an error in the Axis.
3. Power.Enable=TRUE and there is no error in the Axis.
4. MC_Stop.Done AND NOT MC_Stop.Execute.

10.1.3.3 Visualizations

For usage with the PLCopen Library, a set of visualization objects is defined. These visualizations use the placeholder concept, which means that they could be used in an actual visualization several times and be instantiated by replacing the “placeholder” with an effective data structure.

Two types of visualizations exist:

As placeholder, an instance of Axis_Ref should be used. These are named:MC_VISU_Axis_name. Here the name could be state machine or its actual.

As placeholder, an instance of the respective PLCopen Function Block should be used. These visualizations are named MC_VISU_FB_name where "name" could be MoveAbsolute or MoveVelocity, so the complete element is named MC_VISU_FB_MoveAbsolute or MC_VISU_FB_MoveVelocity.

The background color and the color for the title of each element could be changed. The colors are defined in some global predefined variables in MC_VISU_COLOR_INFORMATION. By changing these values, different colors will be used.

MC_VISU_COLOR_INFORMATION (GVL)

InOut:

Name	Type	Initial	Comment
MC_VISU_BACKGROUND_COLOR	DWORD	16#FF888888	16#FFRRGGBB (* Color combination for the Grey color*)
MC_VISU_TITLE_COLOR	DWORD	16#FFFFFF00	16#FFRRGGBB (* Color combination for yellow*)

10.1.3.4 Error codes

Besides the diagnosis information of the drive which is described in the respective drive documentation, there are a numbers of error codes directly related to the Function Blocks. These error codes are displayed at the output “ErrorID” of the Function Block.

Error Code	Mnemonic	Explanation
0	MC_Ok	No Error
1	WRONG_STATE	A Function Block was activated not according to the state machine, e.g. tried to start a movement while in state Disabled.
2	DRIVE_PROBLEM	The drive indicates an error, e.g. tripped.
3	PARAMETER_EXCEEDS_LIMIT	A parameter at the Function Block is outside the possible range. This does not refer to the parameter range which is allowed for the drive but just to the 32-Bit Integer which is used for internal calculation.
4	NO_FIELD_ACCESS	No fieldbus connection to the drive.
5	BUS_PROBLEM	Not used
6	ABS_SWITCH_ERROR	During Homing, (when done by Function Blocks) limit switch not according to moving direction e.g. the positive switch occurred when moving in negative direction.
7	TIMEOUT	Timeout in block execution.
8	NAK	Parameter access not applicable
9	MC_TimeLimitExceeded	Used by Function Blocks with TimeLimit.
10	MC_DistanceLimitExceeded	Used by Function Blocks with DistanceLimit.
11	MC_TorqueLimitExceeded	Used by Function Blocks with TorqueLimit.

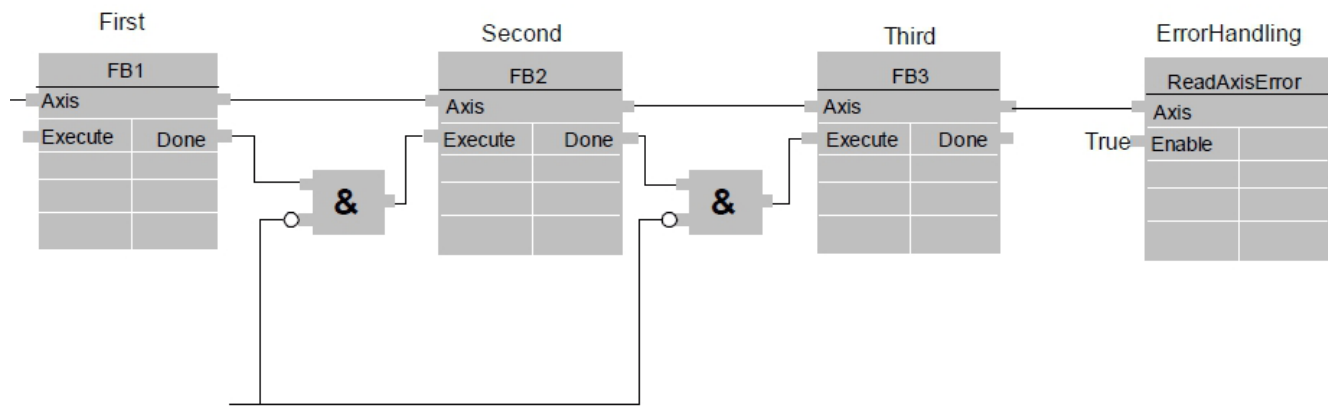
12	NOT_IMPLEMENTED	Functionality not implemented for certain axis type.
101	ErrorID_POSITION_FOLLOW	Following error, caused by > position error =>ERRORSTOP (parameter POS_LAG_PERCENTAGE)
102	ErrorID_POSSW	Positive software limit switch => ERRORSTOP. The actual position did exceed the positive Software limit switch position. This supervision has to be activated with MC_WriteParameter.
103	ErrorID_NEGSW	Negative software limit switch => ERRORSTOP. The actual position did exceed the negative Software limit switch position. This supervision has to be activated with MC_WriteParameter.
104	ErrorID_VELOCITY_FAULT	The measured velocity and commanded velocity are > 50% (related to maximum velocity) apart, for a certain time =>ERRORSTOP (parameter V_CHECKTIME)
105	ErrorID_INTERPOLATION_FAULT	following error, caused by interpolation problem =>ERRORSTOP. Position following error occurred, but reason most likely an interpolation problem, not drive problem (e.g.CAM Table, position step).
110	ErrorID_WARNING_VELOCITYLIMIT	Velocity or acceleration/deceleration are in limitation, set by parameter EnableLimitVelocity, MaxVelocityAppl, MaxDecelerationAppl
111	ErrorID_WARNING_POSITIONLIMITPOS	Position is in limitation towards position limit (SWLimit2DecPos), axis decelerates near positive software limit switch
112	ErrorID_WARNING_POSITIONLIMITNEG	Position is in limitation towards position limit (SWLimit2DecNeg), axis decelerates near negative software limit switch
113	ErrorID_WARNING_POSITIONOVERRUN	A linear axis created a 32bit position overrun(> 2147483647 u=>inc) =>configure modulo
114	ErrorID_WARNING_ABORT	Axis aborted due to too large position gap due to velocity limitation
115	ErrorID_WARNING_MOVEMENT_DIRECTION	Either positive or negative direction blocked by MC_Power

10.1.3.5 Error handling

All access to the drive/motion control is via Function Blocks. Internally these Function Blocks provide basic error checking on the input data. Exactly, how this is done is implementation dependent. For instance, if “MaxVelocity” is set to 6000, and the Velocity input to a Function Block is set to 10,000, a basic error report is generated. In the case where an intelligent drive is coupled via a network to the system, the “MaxVelocity” parameter is stored on the drive. The Function Block must take care of the errors generated by the drive internally. With another implementation, the “MaxVelocity” value could be stored locally. In this case the Function Block will generate the error locally.

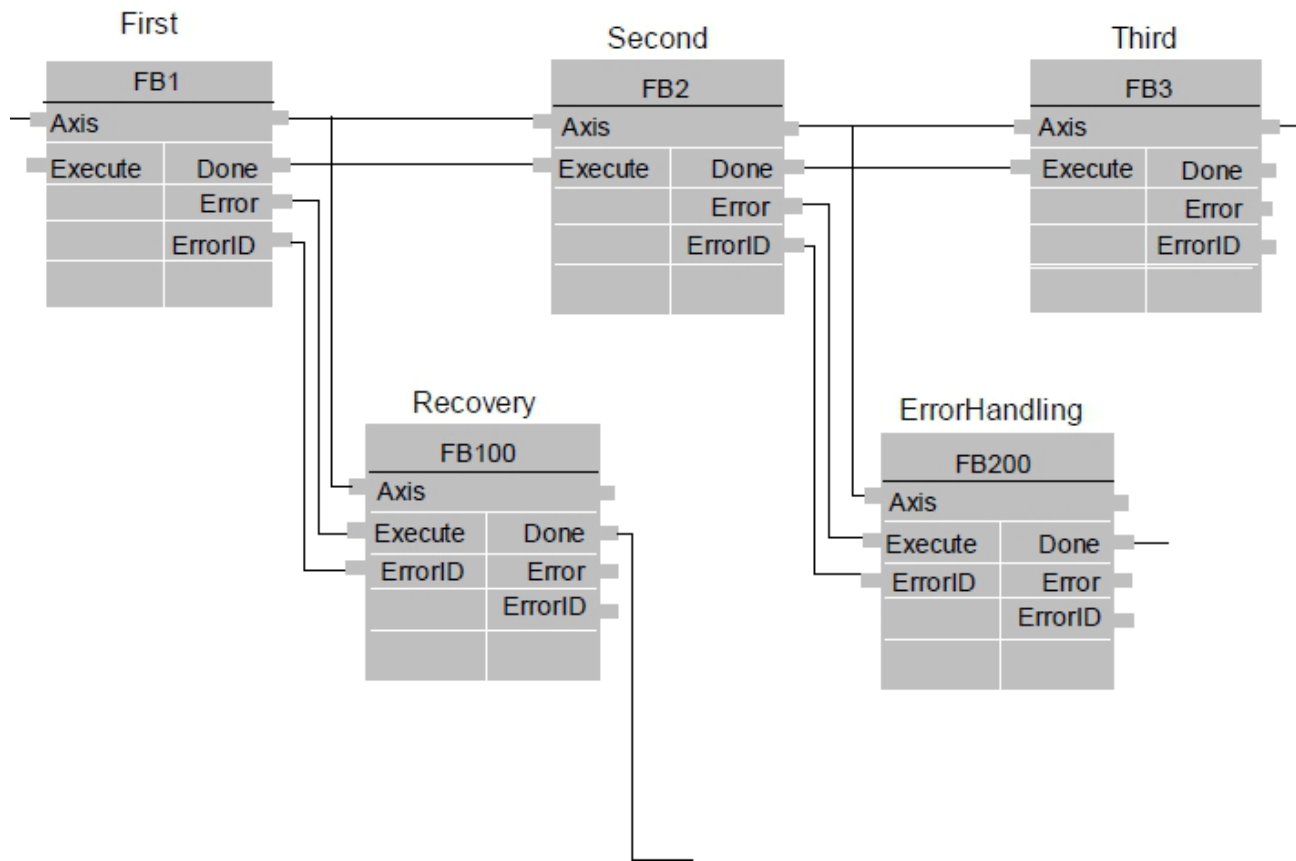
Both centralized and decentralized error handling methods are possible when using the motioncontrol Function Blocks.

Centralized error handling is used to simplify programming of the Function Block. Error reaction is the same independent of the instance in which the error has occurred.



Function Blocks with centralized error handling


Decentralized error handling gives the possibility of different reactions depending on the Function Block in which an error occurred.



Function Blocks with decentralized error handling

10.1.3.6 PLCopen parameter

Additional parameters are available by ReadParameter and WriteParameter Function Blocks.



Note: Following function blocks can be used for the read and write operation. Function-ality of these blocks and its variables are explained in the integrated documentation

MC_ReadParameter

MC_WriteParameter

MC_ReadBoolParameter

MC_WriteBoolParameter

Parameter number (PN)	Name	Comments	Data type	Min.	Max.	Default	R/W
1	CommandedPosition	Commanded position.	DINT				R
2	SWLimitPos	Positive software limit switch position.	DINT	-2147483647	2147483647	2147483647	R/W
3	SWLimitNeg	Negative software limit switch position.	DINT	-2147483647	2147483647	- 2147483647	R/W
4	Enable LimitPos	Enable positive software limit switch.	BOOL	FALSE	TRUE	FALSE	R/W
5	Enable LimitNeg	Enable negative software limit switch.	BOOL	FALSE	TRUE	FALSE	R/W
6	Enable Pos-LagMonitoring	Enable monitoring of position lag (following error).	BOOL	FALSE	TRUE	TRUE	R/W
7	MaxPositionLag	Maximal position lag.	DINT	1	2147483647		R
8	MaxVelocity-System	Maximal allowed velocity of the axis in the motion system.	DINT			32767	R
9	MaxVelocityAppl	Maximal allowed velocity of the axis in the application.	DINT	0**	32767	32767	R/W
10	Actual Velocity	Actual velocity.	DINT	-32767	32767		R
11	CommandedVelocity	Commanded velocity.	DINT	-32767	32767		R
12	MaxAcceleration-System	Maximal allowed acceleration of the axis in the motion system.	DINT			32767	R
13	MaxAccelerationAppl	Maximal allowed acceleration of the axis in the application.	DINT	10	32767	32767	R/W
14	MaxDeceleration-System	Maximal allowed deceleration of the axis.	DINT			32767	R
15	MaxDecelerationAppl	Maximal allowed deceleration of the axis.	DINT	10	32767	32767	R/W
16	MaxJerk	Maximal allowed jerk of axis.	DINT	0*	2147483647	2147483647	R/W
2001	MODULO_NOMINATOR	ABB specific parameter. Used for PLC-based Motion Control implementation: Gearbox modifier to MODULO_RANGE	DINT	1	2147483647	1	R/W
2002	MODULO_DENOMINATOR	ABB specific parameter. Used for PLC-based Motion Control implementation: Gearbox modifier to MODULO_RANGE	DINT	1	2147483647	1	R/W
2003	Enable-Limit2Decelerate	Enable software limit switches to decelerate	BOOL	FALSE	TRUE	FALSE	R/W
2004	EnableLimitAbort	Enable that software limit switches will abort ongoing movement FALSE = Limits position a. velocity, decelerates and shows a warning until the position limit is reached, then ERROR STOP TRUE = Switches off any ongoing motion and decelerates to the position limit, then ERROR STOP	BOOL	FALSE	TRUE	FALSE	R/W
2005	Enable-LimitVelocity	If the velocity is limited the unmoved position will be covered whenever possible	BOOL	FALSE	TRUE	FALSE	R/W
2006	SWLimit2DecPos	Used as end position for EnableLimit2Decelerate	LREAL	-2147483647	2147483647	2147483647	R/W
2007	SWLimit2DecNeg	Used as end position for EnableLimit2Decelerate	LREAL	-2147483647	2147483647	2147483647	R/W
2008	MaxPositionGapLL	Used to stop the ongoing movement if position is behind	LREAL	0	214748364700	0	R/W

0* means: no limitation of jerk is performed.

**Axis will stay in stop.

***is modified by CMC_Axis_Control_Parameter, the max. Value is calculated in increments, the value which is delivered by ReadParameter will be given in [u].

In addition to the above parameters certain other operation can be done using the below parameters from the data type "Axis_Parameter"

Name	Type	Initial	Comment
paraFilterVariant	INT		Filter for actual velocity 0 = PT1 1 = LinearRegression
paraFilterTime	INT	10	Time in PLC cycles, used with para-Filter-Variant
paraFilterForecast	INT	0	Time in PLC cycles, used with para-Filter-Variant = 1
paraReverseDirection	INT	0	Changes the direction for actual and reference positions based on the mode selected. 0 = normal direction 1 = reverse input position 2 = reverse output position and speed reference 3 = reverse both
paraEarlyClosedLoop	BOOL	FALSE	TRUE: hold the position when Drive_Release is set (not wait for Drive_InOperation = TRUE)
paraLateOpenLoop	BOOL	FALSE	TRUE: hold the position until Drive_InOperation = FALSE

10.1.3.7 Limits

Limitations for the inputs of PLCopen Function Blocks when used with CMC_Basic_Kernel

Parameter	Min.	Max.
Velocity	0	x
Acceleration, Deceleration	0	x
Position	-2147483647	2147483647

10.1.3.8 General restrictions

Restrictions for the available function blocks

As buffered mode, MC_Aborting is realized as a default. This does NOT mean that the axis stops when another movement is started while an ongoing movement is still active. It means instead that the new movement will take control immediately and change the velocity to its own velocity by using its own acceleration or deceleration.

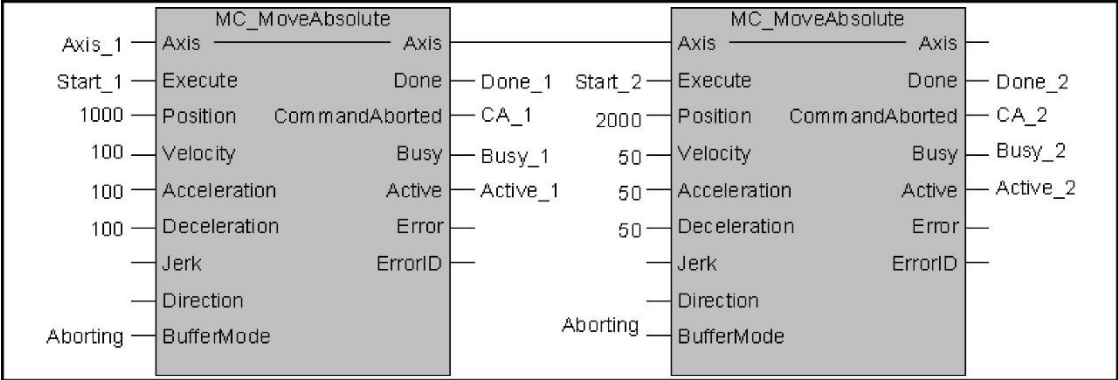
The buffered mode MC_Buffered could be reached with using the axis state StandStill as enable signal for the Execute of the next block.

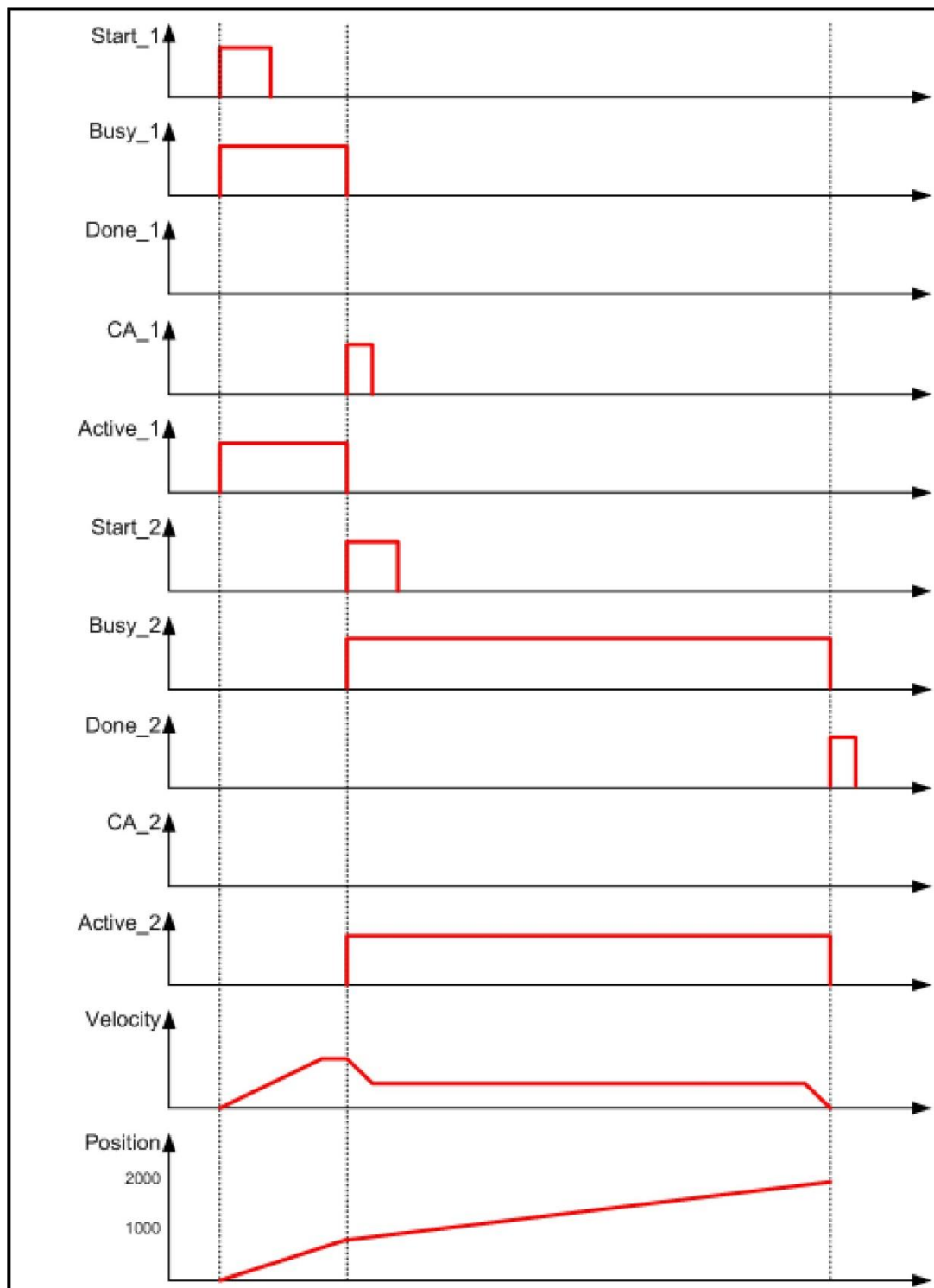
From the Extended Inputs and Outputs at the Function Blocks, the following are not realized:

BufferedMode: The realization just supports the MC_Aborting mode.

The following Outputs at ReadStatus are not supported: ConstantVelocity, Accelerating and Decelerating.

TorqueLimit for Homing Function Blocks.





MC_Aborting Mode

The diagram shows the behavior with BufferMode MC_Aborting, which is the only available BufferMode. When the second Block is activated, it will take control and will continue on its own velocity. The velocity is changed by using the acceleration value from the second Function Block. The movement will not be stopped in between. The first Function Block shows CommandAborted when the second Function Block is activated.

MC_Buffered

A behavior according to BufferMode MC_Buffered could be reached by using the Done output from the first Function Block to enable the Execute of the second Function Block.

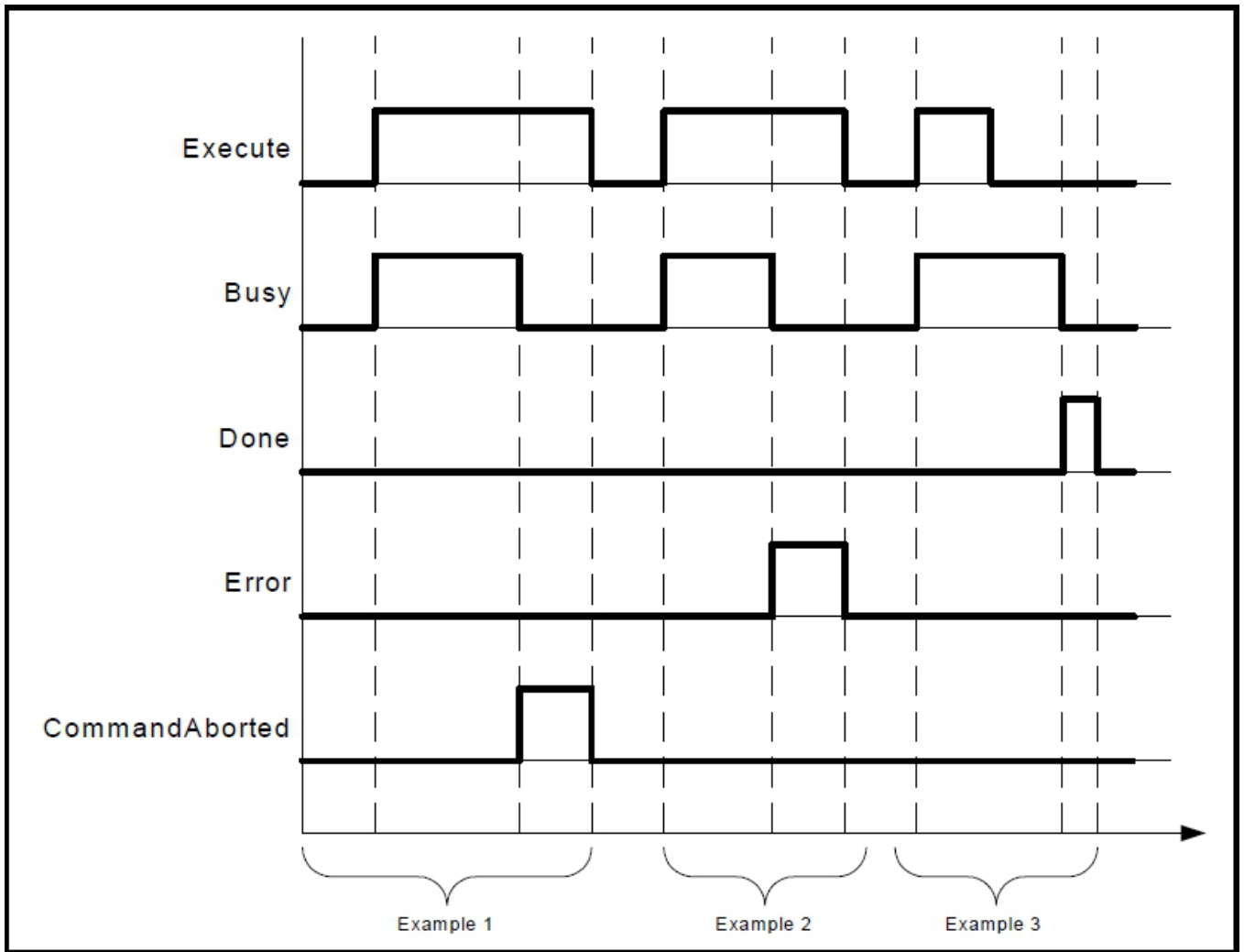
10.1.3.9 Behavior of the function block inputs and outputs

10.1.3.9.1 General rules

General rules

Output exclusivity	The outputs Busy, Done, Error, and CommandAborted are mutually exclusive: Only one of them can be TRUE on one Function Block. If Execute is TRUE, one of these outputs has to be TRUE. Only one of the outputs Active, Error, Done and CommandAborted is set at the same time.
Output status	The outputs Done, InGear, InSync, InVelocity, Error, ErrorID and CommandAborted are reset with the falling edge of Execute. However, the falling edge of Execute does not stop or even influence the execution of the actual Function Block. It must be guaranteed that the corresponding outputs are set for at least one cycle if the situation occurs, even if execute was reset before the Function Block completed. If an instance of a Function Block receives a new execute before it has finished (as a series of commands on the same instance), the Function Block will not return any feedback, like Done or CommandAborted, for the previous action.
Input parameters	The parameters are used with the rising edge of the execute input. To modify any parameter, it is necessary to change the input parameter(s) and to trigger the motion again.
Missing input parameters	According to IEC 61131-3, if any parameter of a Function Block input is missing (open) then the value from the previous invocation of this instance will be used. In the first invocation the initial value is applied.
Position versus distance	Position is a value defined within a coordinate system. Distance is a relative measure related to technical units. Distance is the difference between two positions.
Sign rules	Velocity, Acceleration, Deceleration and Jerk are always positive values. Position and Distance can be both positive and negative.
Error Handling Behavior	All Function Blocks have two outputs, which deal with errors that can occur while executing that Function Block. These outputs are defined as follow:
	Error: Rising edge of Error informs that an error occurred during the execution of the Function Block.
	ErrorID: Error number

	<p>The outputs Done, InVelocity, InGear, and InSync mean successful completion, so these signals are logically exclusive to Error.</p> <p>Types of errors:</p> <ul style="list-style-type: none"> • Function Blocks (e.g. parameters out of range, state machine violation attempted), • Communication, <p>Drive Instance errors do not always result in an axis error (bringing the axis to StandStill). The error outputs of the relevant Function Block are reset with falling edge of Execute.</p>
Function Block Naming	In case of multiple libraries within one system (to support multiple drive/ motion control systems), the Function Block naming may be changed to MC_FunctionBlockName_SupplierID.
Behavior of Done output	<p>The outputs Done, InGear, InSync... are set when the commanded action has been completed successfully. With multiple Function Blocks working on the same axis in a sequence, the following applies:</p> <p>When one movement on an axis is interrupted with another movement on the same axis without having reached the final goal, Done of the first Function Block will not be set.</p>
Behavior of CommandAborted output	CommandAborted is set, when a commanded motion is interrupted by another motion command. The reset-behavior of CommandAborted is like that of Done. When CommandAborted occurs, the other output-signals such as InVelocity are reset.
Inputs exceeding application limits	If a Function Block is commanded with parameters which result in a violation of application limits, the instance of the Function Block generates an error. The consequences of this error for the axis are application specific and thus should be handled by the application program.
Behavior of Busy output	<p>Every Function Block can have an output Busy, reflecting that the Function Block is not finished. Busy is SET at the rising edge of Execute and RESET when one of the outputs Done, Aborted, or Error is set. It is recommended that this Function Block should be kept in the active loop of the application program for at least as long as Busy is true, because the outputs may still change. For one axis, several Function Blocks might be busy, but only one can be active at a time.</p> <p>Exceptions are MC_SuperImposed and MC_Phasing, where more than one Function Block related to one axis can be active.</p>
Output Active	The output Active is required on buffered Function Blocks. This output is set at the moment the Function Block takes control of the motion of the according axis. For unbuffered mode the outputs Active and Busy can have the same value.
Enable and Valid/Status	The input Enable is coupled to output Valid. Enable is level sensitive, and Valid shows that a valid set of outputs is available at the Function Block. The output Valid is TRUE as long as an output value of Valid is available and the input Enable is TRUE. The relevant output value can be refreshed as long as the input Enable is TRUE. If there is a Function Block error, the output is not Valid (Valid set to FALSE). When the error condition disappears, the values will reappear and output Valid will be set again.

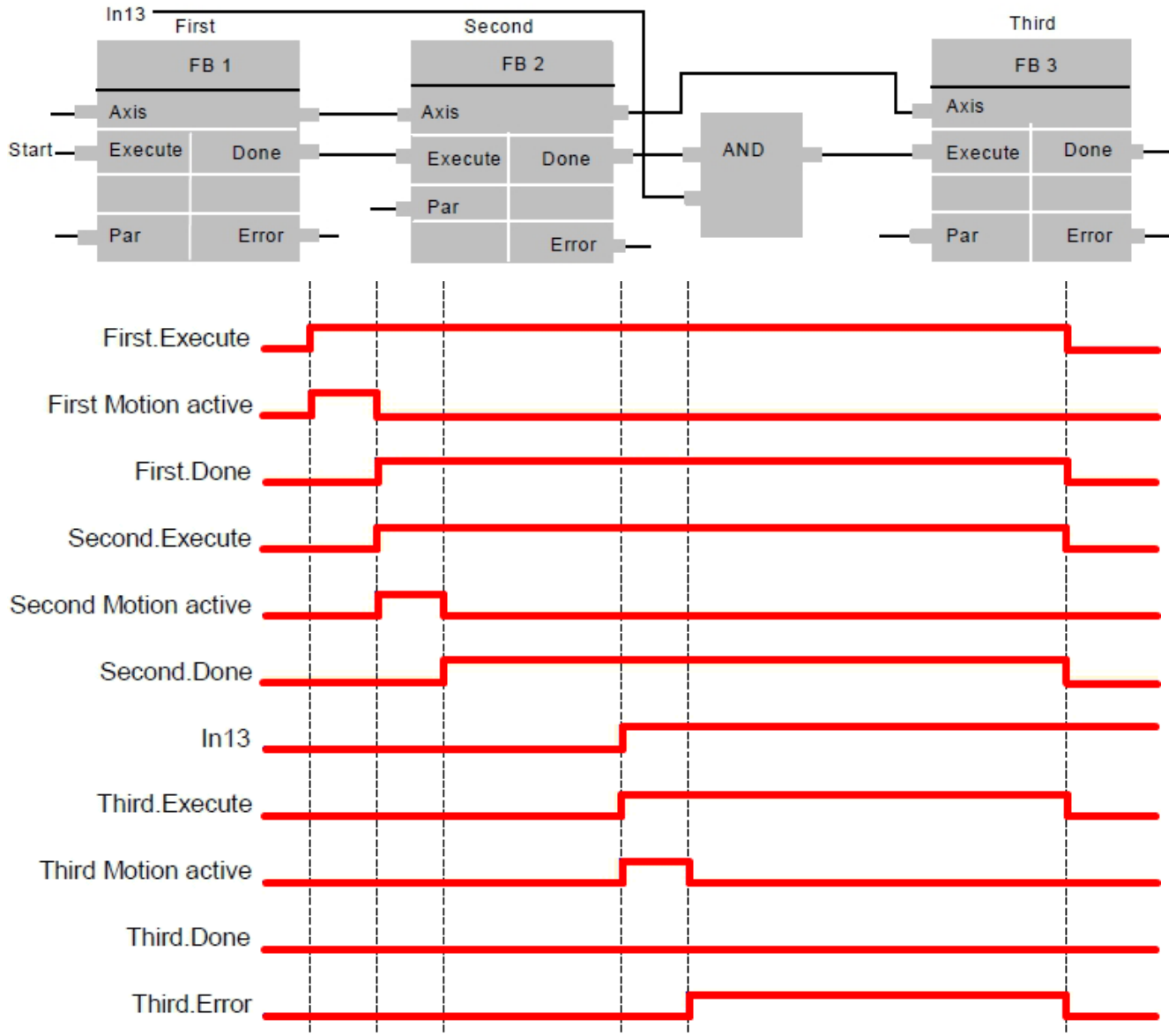


Behavior of the Execute/Done style Function Blocks.

10.1.3.9.2 Why is the command input edge sensitive?

The input Execute for the different Function Blocks described in this document always triggers the function with its rising edge. The reason for this is that with edge triggered Execute new input values may be commanded during execution of a previous command. The advantage of this method is a precise management of the instant a motion command is performed. Combining different Function Blocks is then easier in both centralized and decentralized models of axis management. The output Done can be used to trigger the next part of the movement. The example given below is intended to explain the behavior of the Function Block execution.

The following figure illustrates the sequence of three Function Blocks First, Second and Third controlling the same axis. These three Function Blocks could be for instance various absolute or relative move commands. When First is completed the motion its rising output "First.Done" triggers "Second.Execute". The output "Second.Done" AND "In13" triggers the "Third.Execute".



Function Blocks to perform a complex movement

10.1.3.9.3 The input ContinuousUpdate

As described in the previous chapter, the input Execute triggers a new movement. With a rising edge of this input the values of the other Function Block inputs define the movement. Until version 1.1 of PLCopen there was the general rule that a later change in these input parameters does not affect the ongoing motion.

Nevertheless, there are numerous application examples, where a continuous change of the parameters is needed. The user could retrigger the input Execute of the Function Block, but this complicated the application.

Therefore, the input ContinuousUpdate has been introduced. It is an extended input to all applicable Function Blocks. If it is TRUE, when the Function Block is triggered (rising Execute), it will

- as long as it stays TRUE – make the Function Block use the current values of the input variables and apply it to the ongoing movement. This does not influence the general behavior of the Function Block nor does it impact the single axis state diagram. In other words, it only influences the ongoing movement and its impact ends as soon as the Function Block is no longer Busy or the input ContinuousUpdate is set to FALSE.



Note: It can be that certain inputs like BufferMode are not really intended to change every cycle. However, this has to be dealt with in the application, and is not forbidden in the specification

If ContinuousUpdate is FALSE with the rising edge of the input Execute, a change in the input parameters is ignored during the whole movement and the original behavior of previous versions is applicable. The ContinuousUpdate is not a retriggering of the input Execute of the Function Block. A retriggering of a Function Block which was previously aborted, stopped, or completed, would regain control on the axis and modify its single axis state diagram. Opposite to this, the ContinuousUpdate only effects an ongoing movement. Also, a ContinuousUpdate of relative inputs (e.g. Distance in MC_MoveRelative) always refers to the initial condition (at rising edge of Execute).

Example

MC_MoveContinuousRelative is started at Position 0 with Distance 100, Velocity 10 and ContinuousUpdate set TRUE. Execute is Set and so the movement is started to position 100.

While the movement is executed (let the drive be at position 50), the input Distance is changed to 130, Velocity 20.

The axis will accelerate (to the new Velocity 20) and stop at Position 130 and set the output Done and does not accept any new values

10.1.3.10 Unit of length

The only specification for physical quantities is made on the unit of length (noted as [u]) that is to be coherent with its derivatives i.e. (velocity [u/s]; acceleration [u/s²]; jerk [u/s³]). Nevertheless, the unit [u] is not specified (manufacturer dependent). Only its relations with others are specified.

10.1.3.11 Aborting versus buffered modes

Some of the Function Blocks have an input called BufferMode. With this input, the Function Block can either work in a Non-buffered mode (default behavior) or in a Buffered mode. The difference between those modes is when they should start their action:

- A command in a non-buffered mode acts immediately, even if this interrupts another motion,
- A command in a buffered mode waits till the current Function Block sets its output Done (or InPosition, InVelocity...).

The library just supports the mode "aborting" (MCAborting)

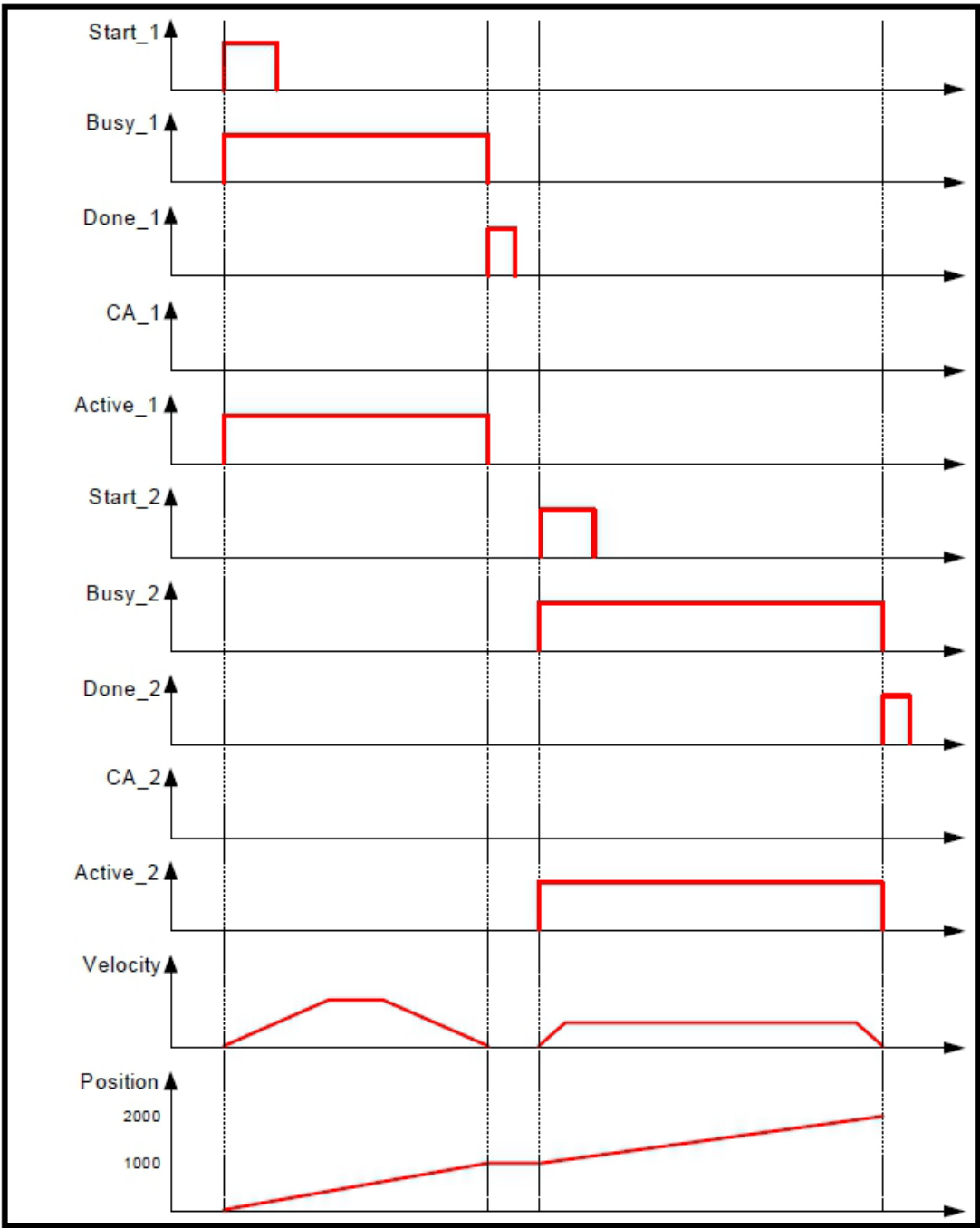
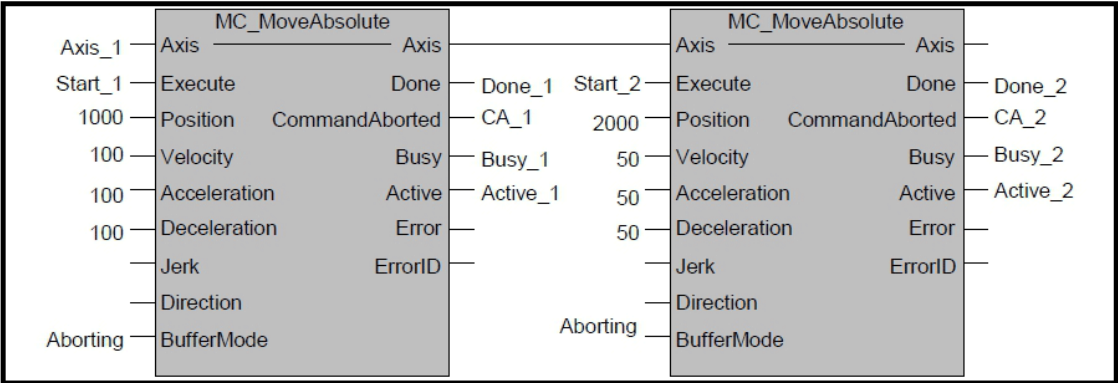
If an on-going motion is aborted by another movement, it can occur that the braking distance is not sufficient due to deceleration limits.

In rotary axis, a modulo can be added. A modulo axis could go to the earliest repetition of the absolute position specified, in cases where the axis should not change direction and reverse to attain the target position.

In linear systems, the resulting overshoot can be resolved by reversing, as each position is unique and therefore there is no need to add a modulo to reach the correct position.

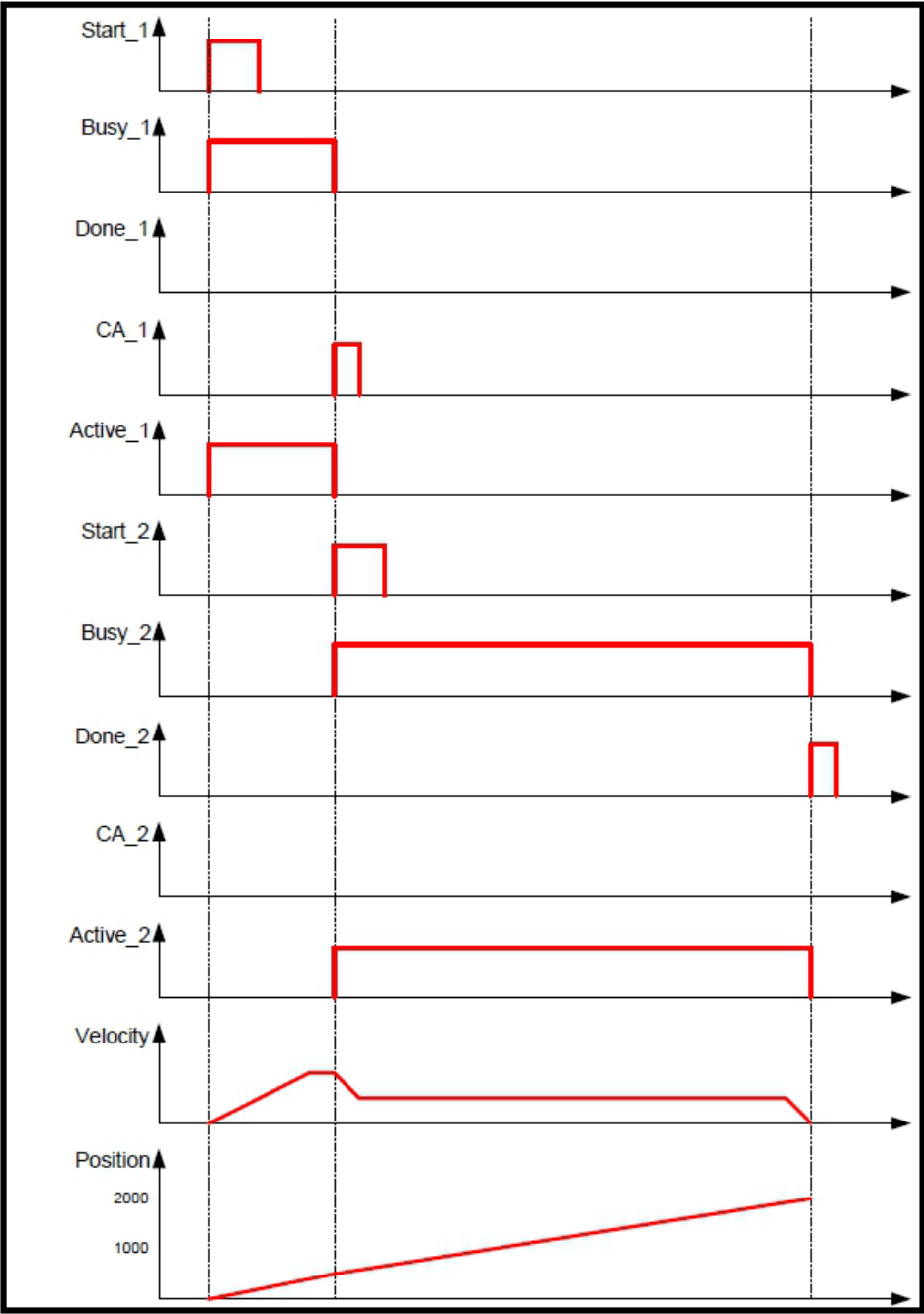
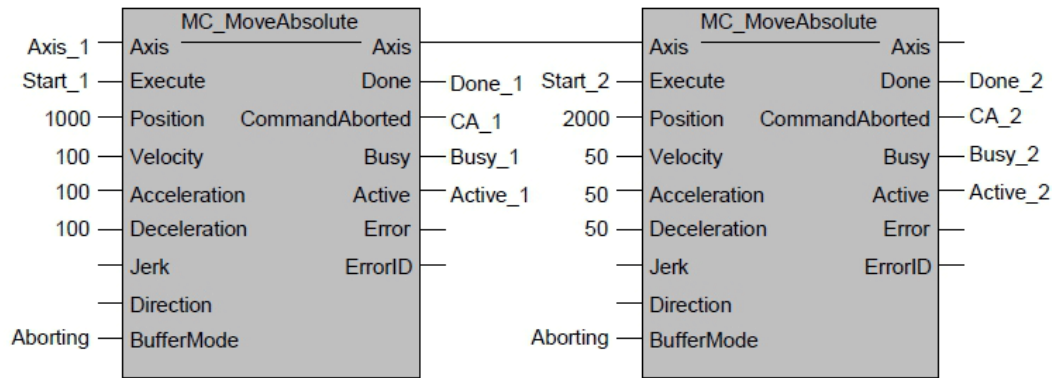
The following examples describe the different behavior of these modes:

Example 1:Standard behavior of two following absolute movements



Timing diagram for example above without interference between Function Block 1 and FunctionBlock 2

Example 2: Aborting motion

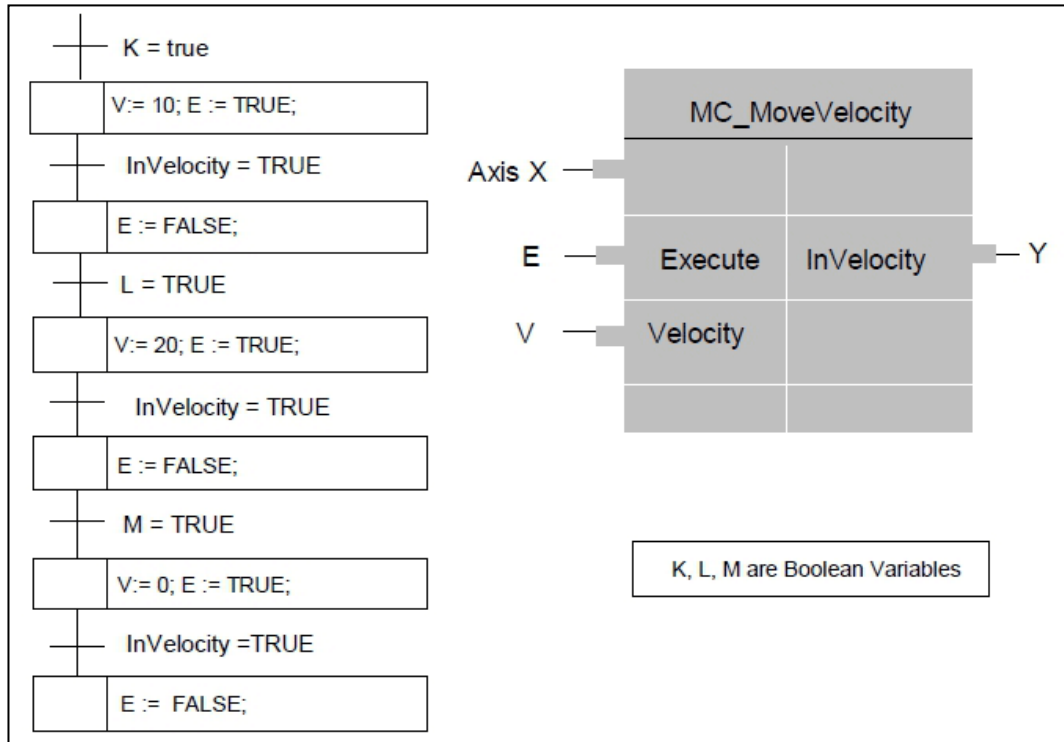


Timing diagram for example above with Function Block 2 interrupting Function Block 1 (McAborting Mode)

10.1.3.12 PLCopen Examples

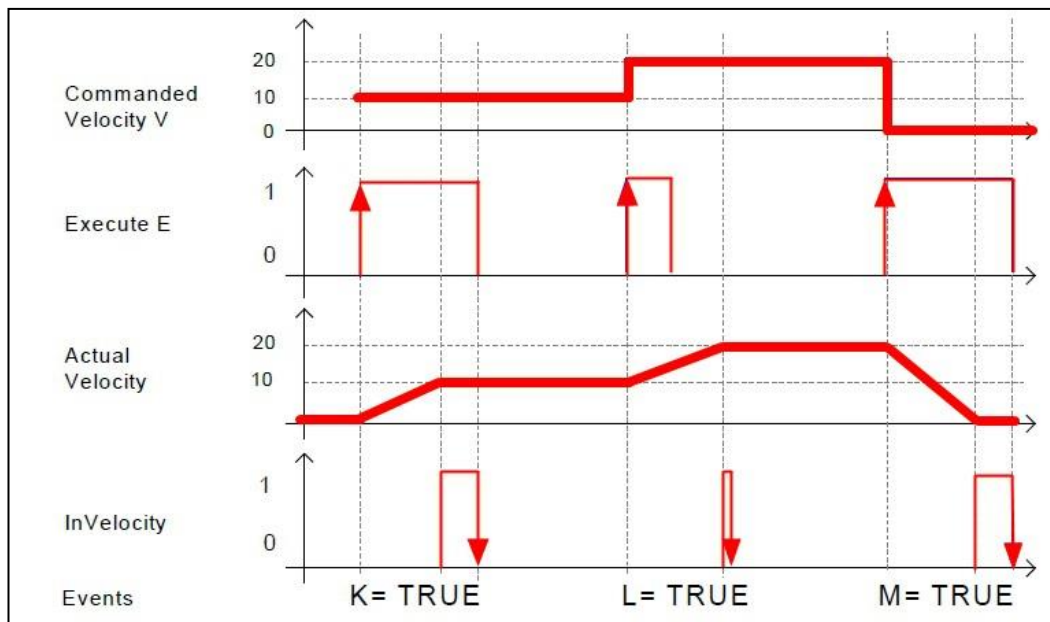
Example: A function block instance controls different motions of an axis

The following figure shows an example where the Function Block (MC_MoveVelocity) is used to control AxisX with three different values of Velocity. In a Sequential Function Chart (SFC) the velocity 10, 20, and 0 is assigned to V. To trigger the input, execute with a rising edge the variable E is stepwise set and re-set.



Single Function Block with SFC

The following timing diagram explains how it works:



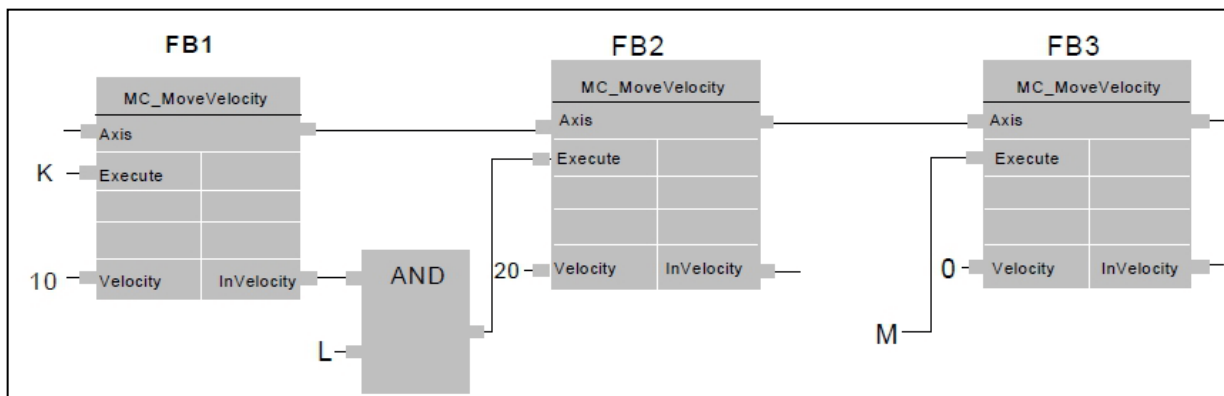
Timing diagram for a usage of single Function Block



Note: The second InVelocity is set for only one cycle because the Execute has gone low before the ActualVelocity equals CommandedVelocity.

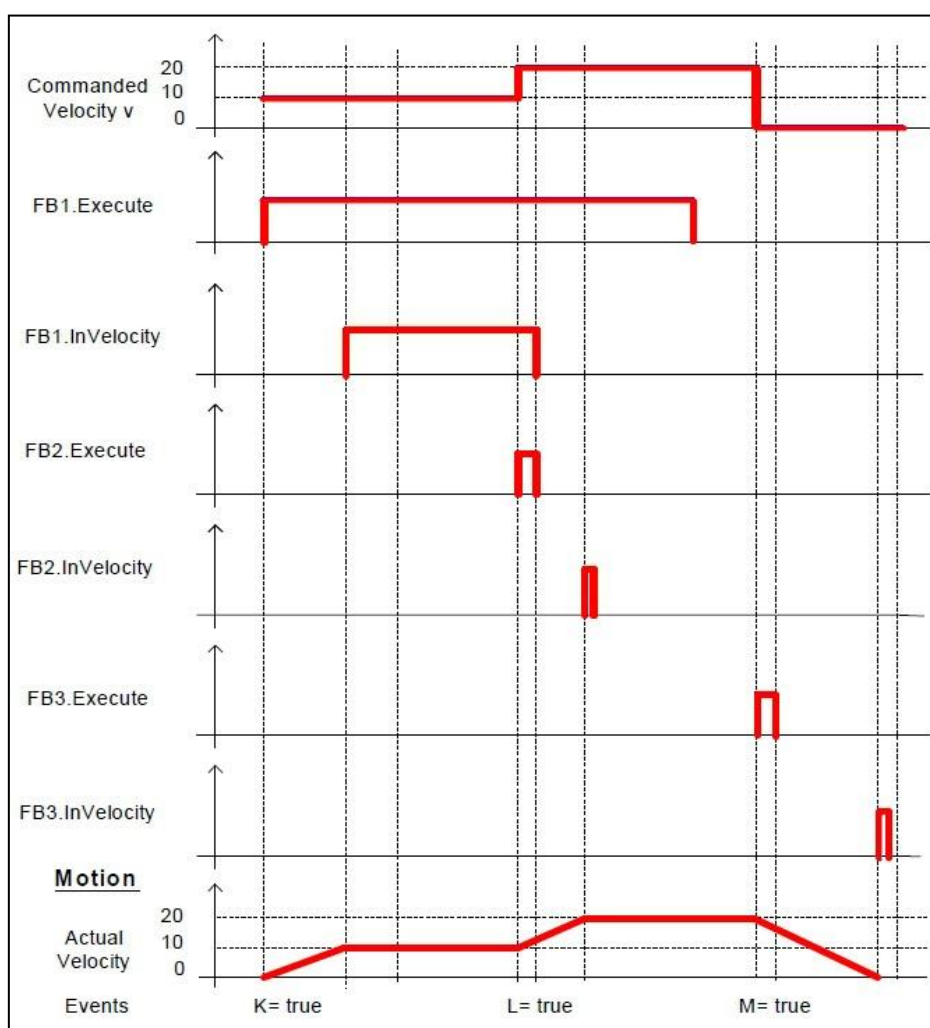
Example: Different function block instances control the motions of an axis

Different instances related to the same axis can control the motions on an axis. Each instance will then be responsible for one part of the global profile.



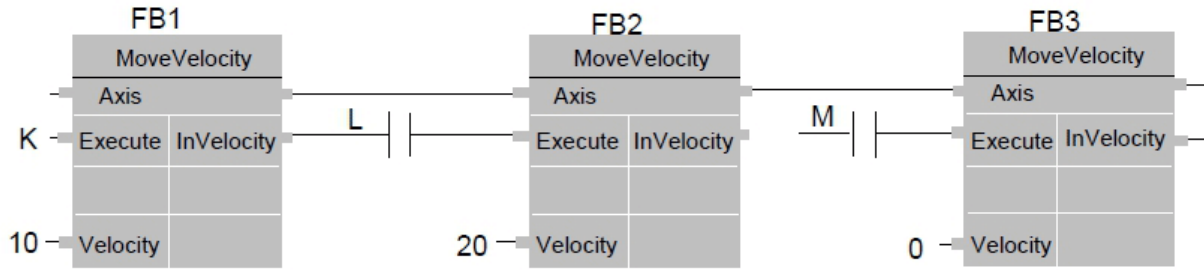
Cascaded Function Blocks

The timing diagram:



Cascaded Function Blocks timing diagram

A corresponding solution written in LD looks like:



Cascaded Function Blocks with LD

10.1.4 PLC-based motion control

10.1.4.1 PLC-based motion control architecture

With PS5611-Motion different Motion Control system structures are possible. Independent of the system structure a typical Motion Control application consists of the following system elements:

An application program which contains PLCopen Function Blocks that defines the general application behavior and logic.

A profile generator which generates a position profile based on the dynamic specifications of the application program to guide the axis to the desired positions.

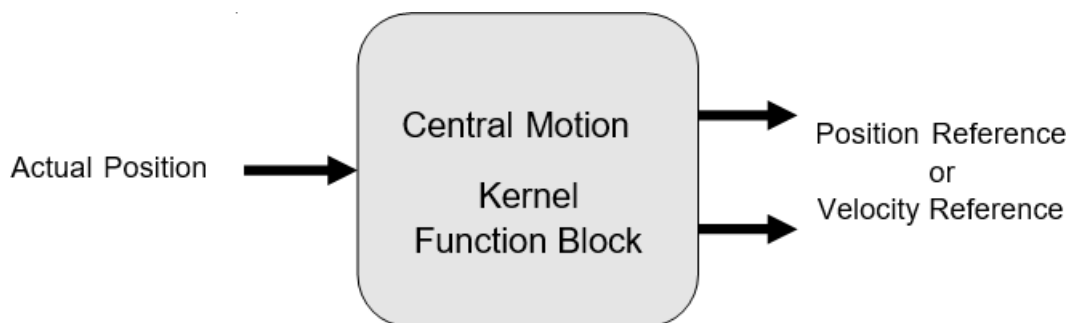
A position control loop which outputs a speed reference signal to minimize the following error.

To achieve the best system structure for an application these components can be separated into different devices. Each type of structure has its own kind of interface and type of signals which need to be transferred between the interacting devices.



Note: All shown Motion Control system structures (Central Motion Control with or without position control loop) can be combined together in the same application program for a Motion Control project.

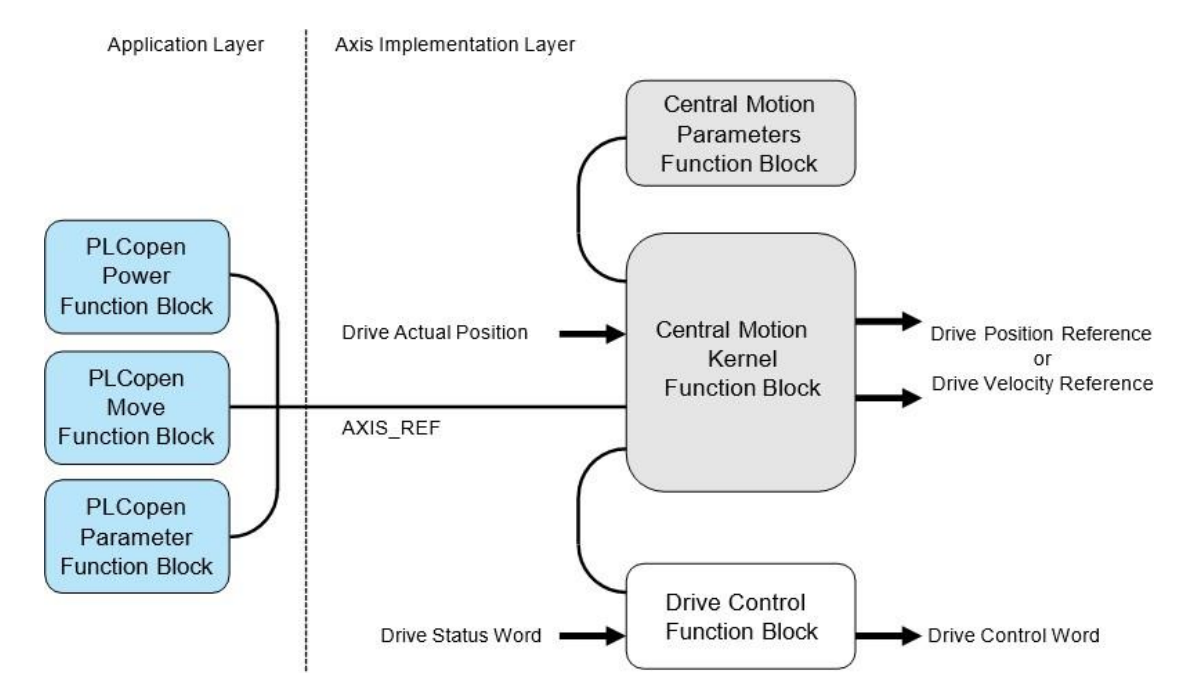
With the Function Blocks of motion library, a Motion Control profiler can be used inside the PLC. As shown in the following figure it is needed to provide the actual position of the drive. The output can be either a position or a velocity reference signal. The used output signals will then be used to move the axis in the desired way.



There are 2 possibilities to send a reference value to the drive:

When the position control loop is closed by the PLC by a CMC_Basic_Kernel FunctionBlock, the output Speed_Reference should be connected to the drive. The value of Speed_Reference can be scaled with the axis parameters Max_Rpm and Ref_Max.

When the position control loop is closed by the drive, the output Position_Reference should be connected to the drive. The unit for the output Position_Reference is incremented as well as the input Drive_Actual-Position.



Architecture for centralized Motion Control

In general, the programming of a machine consists of two layers as shown in the figure above.

In the application layer Function Blocks according to PLCopen Motion Control are used to program the application sequences with all necessary types of movements and administrative commands. Due to the standard PLCopen Motion Control this can be reused in any other machine programs that use PLCopen Function Blocks.

The axis implementation layer is responsible for the execution of the commands from the application layer and can be programmed for each axis in a different way depending on the hardware components used.

Needed function blocks for an application with PLC-based Motion Control

Library	Content
ABB_MotionControl_AC500.library	Kernel Function Block, Parameters FunctionBlock, Axis Simulation Function Block
	Data types for AC500 Motion Control
	Motion Control Function Blocks according to PLCopen




Note: For a central motion axis implementation the use of the Function Blocks CMC_Basic_Kernel and CMC_Axis_Control_Parameter are mandatory.

The library design is independent from any bus architecture or any specific drive features.

Example system architecture is shown below.

System	Velocity reference	Position feedback
System A	Output via analog output channel as voltage or current	From incremental encoder connected to CD522 IO module
System B	Output via EtherCAT network	Input via EtherCAT network
System C	Output as frequency signal of CD522 IO module	From incremental encoder connected to CD522 IO module
System D	Output via PROFINET IO network	Input via PROFINET IO network
System E	Output via PTO & PWM channel in eCo V3	Input via either encoder (using onboard IO), or the PTO or PWM pulse count.

In case the velocity reference value is used from the kernel Function Block the position control loop is closed inside the PLC. In this case, it is necessary to adjust the related parameters from the parameters Function Block. When the position reference will be used the position control loop is closed inside the drive. In this case, the internal control loop is just used to monitor the position and velocity.



Note: When the position reference is used for the drive the following aspects have to be taken care of:

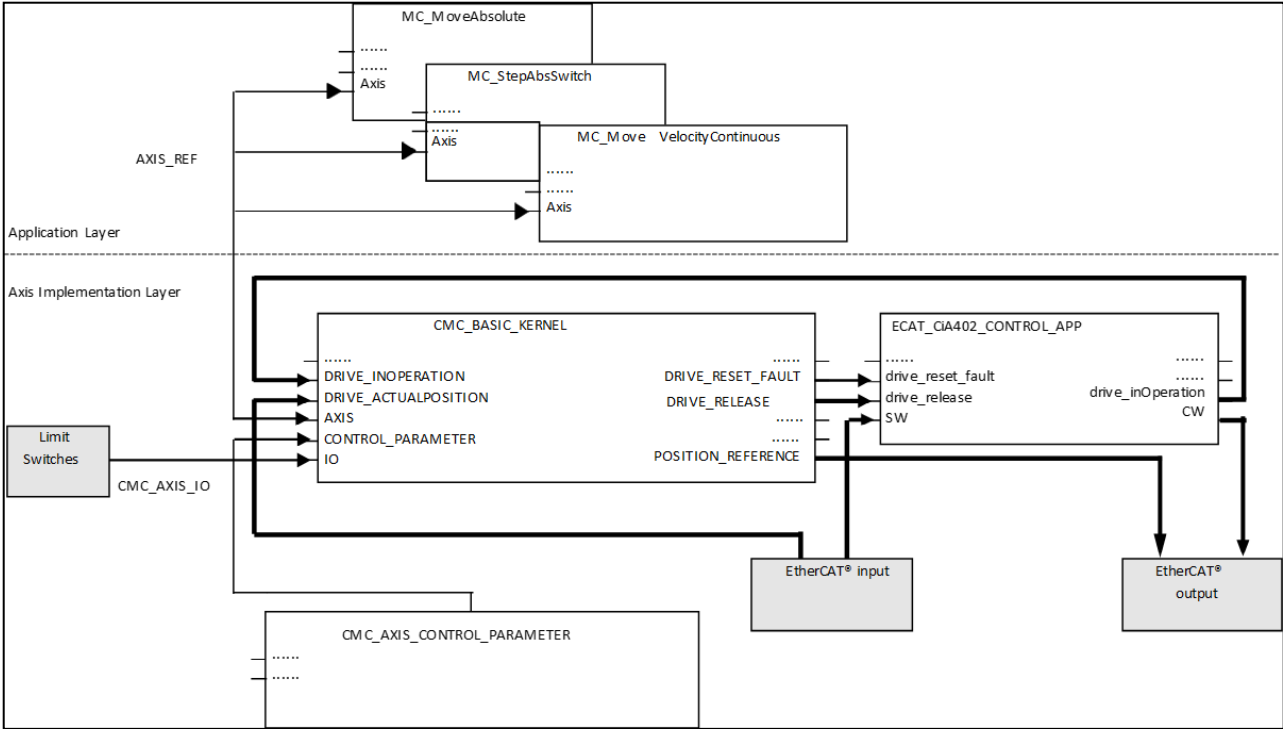
- It is necessary to use a real time fieldbus, like EtherCAT.
- The PLC cycle must be synchronized to the fieldbus cycle.
- The task calculation times may not exceed the used cycle time.

The drive’s status should be managed by a specialized Function Block that supports the used type of drive as shown in the figure above. The kernel Function Block is the main Function Block which is needed to operate an axis with PLC-based Motion Control. It must be used with the parameter Function Block which is the interface to input parameters which are used to setup the axis.



The drive must be accessed outside the CMC_Basic_Kernel Function Block. Actual values and reference values might be transferred by a synchronized fieldbus or by I/O. The FunctionBlock CMC_Basic_Kernel must be called every cycle and at least once before any Function Block MC or MCA is activated.

The following figure shows an example with a CiA402 drive on an EtherCAT network. The maindata signals are drawn in bold lines. Here, the drive will receive a position reference signal which means that the position control loop is closed inside the drive.



In this example the main signals are to be transferred via EtherCAT network. The drive control Function Block “EAT_CiA402_CONTROL_APP” can be found in the ABB_Ecat_CiA402_AC500.library.

If using the eCo V3 PLCs, use the OBIO_PTOMotionKernel function block (separate library ABB_MotionControlEco_AC500.library) instead of CMC_Basic_Kernel for the PTO functionality.



In the AC500 eCo V3 PLC, if PWM is used in the configuration, use the kernel function block OBIO_PWMMotionKernel function block instead of CMC_Basic_Kernel function block.

10.1.4.1.1 Kernel function block

The “KERNEL” Function Blocks are available in two variants.

- The OBIO_PTOMotionKernel / OBIO_PWMMotionKernel function blocks are solely to be used in eCo V3 CPUs and to make use of the integrated stepper-IO. It connects automatically to the internal IOs. Use the PTO or PWM specific kernel block based on your configuration.
- The CMC_Basic_Kernel block is designed to be used in any V3 PLCs and can either work with drives connected to a fieldbus or IOs.

Topic	OBIO_PTOMotionKernel/ OBIO_PWMMotionKernel	CMC_Basic_Kernel
Recommended PLC	eCo V3 PLC	All V3 PLC's

10.1.4.2 Basic functionalities

10.1.4.2.1 How to connect a drive

The connection to a drive must be done with the inputs and outputs of the Function Block CMC_Basic_Kernel. All inputs and outputs of the kernel Function Block with the prefix “Drive_” are intended to be used with a drive, but in some cases not all of them are needed. In all cases the input Drive_ActualPosition has to be connected with the actual position of the axis. This value can be received by an IO module of the PLC or via a fieldbus.

Depending on which device closes the position control loop either the output Speed_Reference or Position_Reference output has to be used. The value of Speed_Reference can be connected to an analog output module or be transferred via a fieldbus. The value of Position_Reference should be exclusively sent via a real-time fieldbus like EtherCAT.

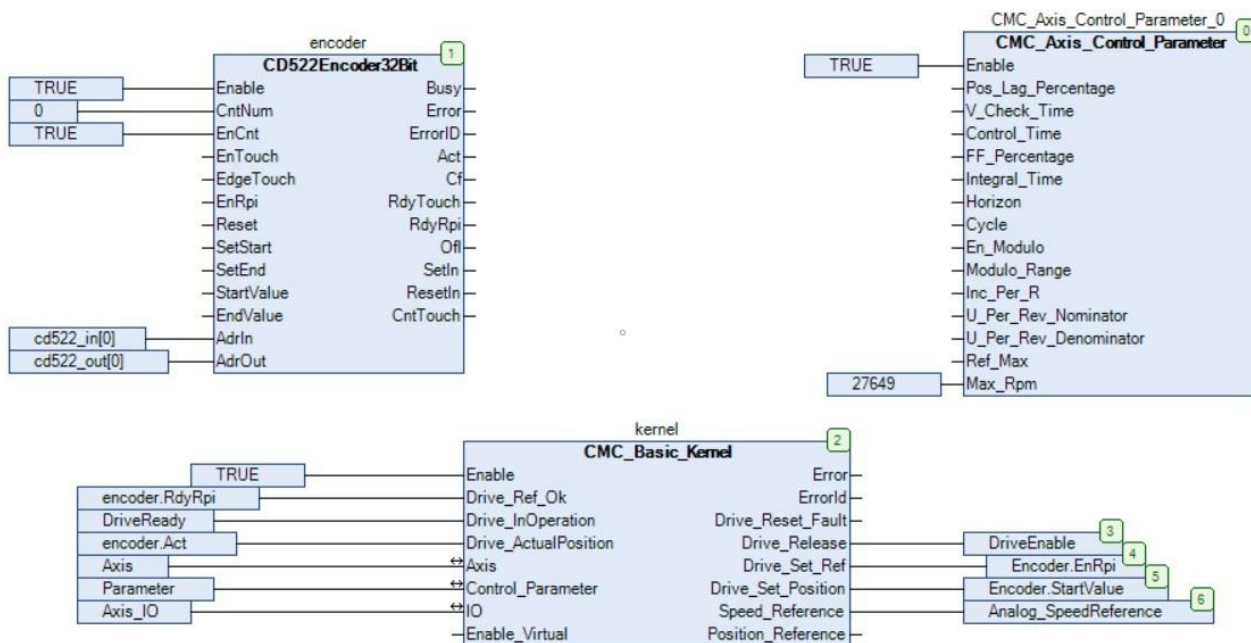
Example 1: Analog drive - Motor with incremental encoder

In this example the position control loop will be closed by the PLC, therefore the input Drive_ActualPosition and the output Speed_Reference are to be used.

In combination with the IO module CD522 and the corresponding Function Block CD522Encoder32Bit the position of the encoder can be used. For the effective resolution of the encoder parameter Inc_Per_R of the parameter Function Block must be used.

The output Speed_Reference can be written directly to the global variable of an output channel of an analog module but can also be transferred via a fieldbus. The scaling of this output value can be done with the parameters Ref_Max and Max_Rpm of the Function Block CMC_Axis_Control_Parameter_Real.

The scaling of the Speed_Reference value can be set with the inputs Ref_Max and Max_Rpm of the parameter Function Block.



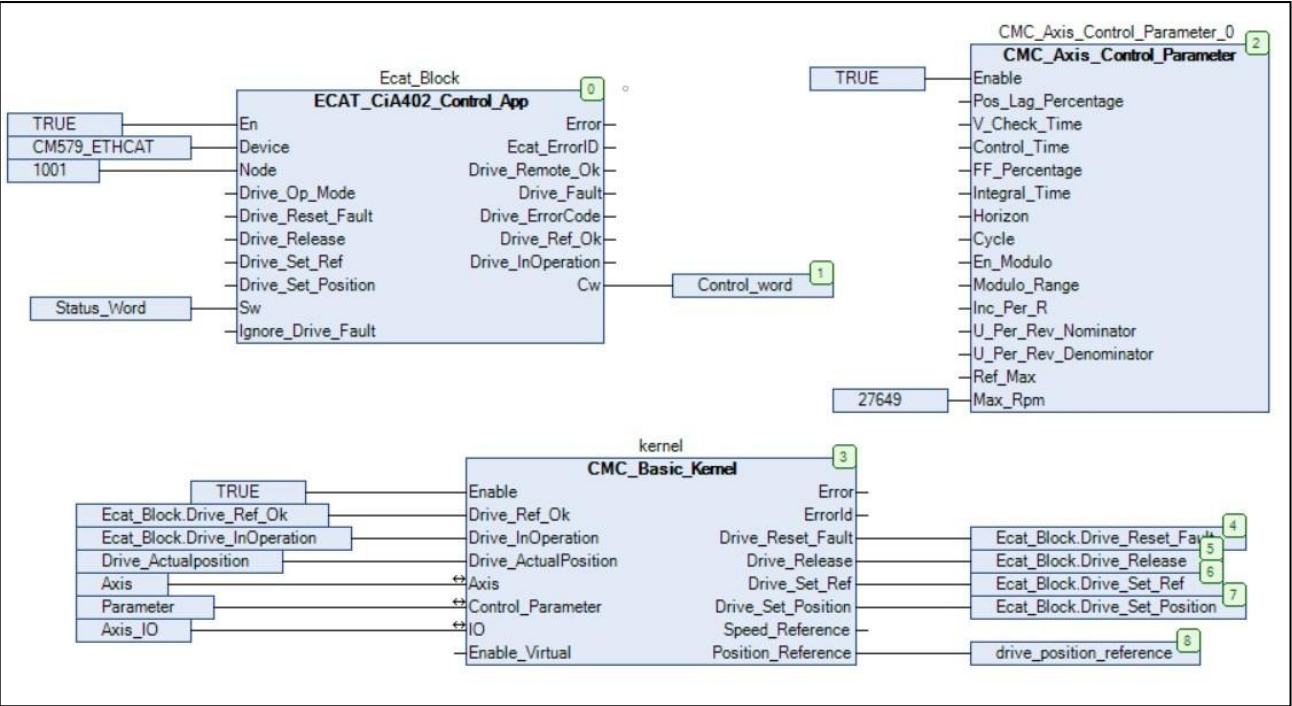
In order to finish a homing sequence which is done by the Function Block MC_StepRefPulse the outputs Drive_Set_Ref and Drive_Set_Position from the kernel Function Block have to be connected with the inputs EN_RPI and START_VALUE of the CD522 IO module Function Block. Also the output RdyRpi of the CD522 IO module Function Block has to be connected with Drive_Ref_Ok from the kernel Function Block.

To enable and disable the drive Drive_Release could be connected to a binary output to activate the drive. Drive_InOperation could be connected to a binary input to get the information that Drive_Release was successful.

Example 2: Servo Drive - Microflex e190 via EtherCAT in continuous positioning mode (csp)

In this example the position control loop will be closed by the drive, therefore the input Drive_ActualPosition and the output Position_Reference are to be used. The inputs referring to the position control loop of the parameter Function Block do not have to be set.

For the effective resolution of the motor's encoder parameter Inc_Per_R of the parameterFunction Block has to be adjusted.

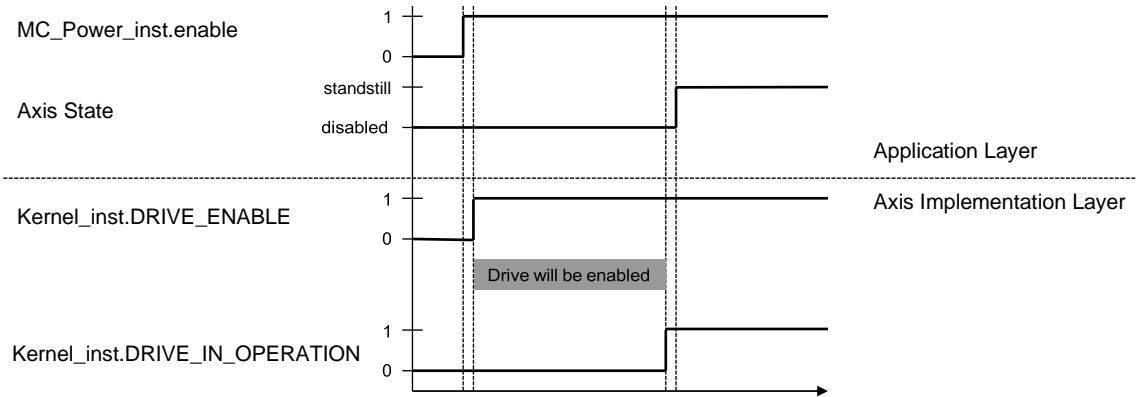


To enable and disable the drive Drive_Release and Drive_Inoperation have to be connected to the control Function Block ECAT_CiA402_Control_App of the library ABB_Ecat_CiA402_AC500.library, which controls the status and control word of the drive.

All Function Blocks from this library are not password protected and free to be changed in order to be adapted for different drives. The library and the Function Blocks are marked with the ending _APP.

10.1.4.2.2 How to enable and disable a drive

To enable a drive the Function Block MC_Power must be used within the applicational layer. The kernel Function Block will then, if possible, output a rising edge on the output Drive_Release which can be connected to the drive-control Function Block which performs the needed actions on the drives control word to enable the drive. As soon as the drive states enabled, this signal can be connected to the input Drive_InOperation of the kernel Function Block. The axis state according to the single axis state diagram of PLCopen will then switch from Disabled to Standstill.



Enabling sequence of a drive

If the drive is in state Disabled or ErrorStop the input Drive_Actual_Position will be copied to the output Position_Reference of the kernel Function Block. The output Speed_Reference will be zero.

When the axis is in operation, which means it is not in state Disabled or ErrorStop, then the output Position_Reference will be calculated by the kernel Function Block and the position control loop will be closed, which outputs nonzero value for the output Speed_Reference in case of a following error. The input Actual_Position should then follow the position reference. The difference of both values is the following error and will be supervised by the kernel Function Block.

In case of drive problem, Drive_InOperation should be reset. The Function Block will open the position control loop and Speed_Reference will be set to zero.

For the most drives the status is control by the drives control word whereas the drives status word represents its actual status. In order to enable the drive it might be necessary to pass through several drives states according a defined scheme which depends on the used drive. Therefore the library ABB_Ecat_CiA402_AC500.library is added to PS5611-Motion package which contains Function Blocks to operate with different drives on an EtherCAT network. There is also the PS5605-DRIVES library package which can be used to control the state of other ABBdrives and other protocols.

10.1.4.2.3 How to use the axis simulation

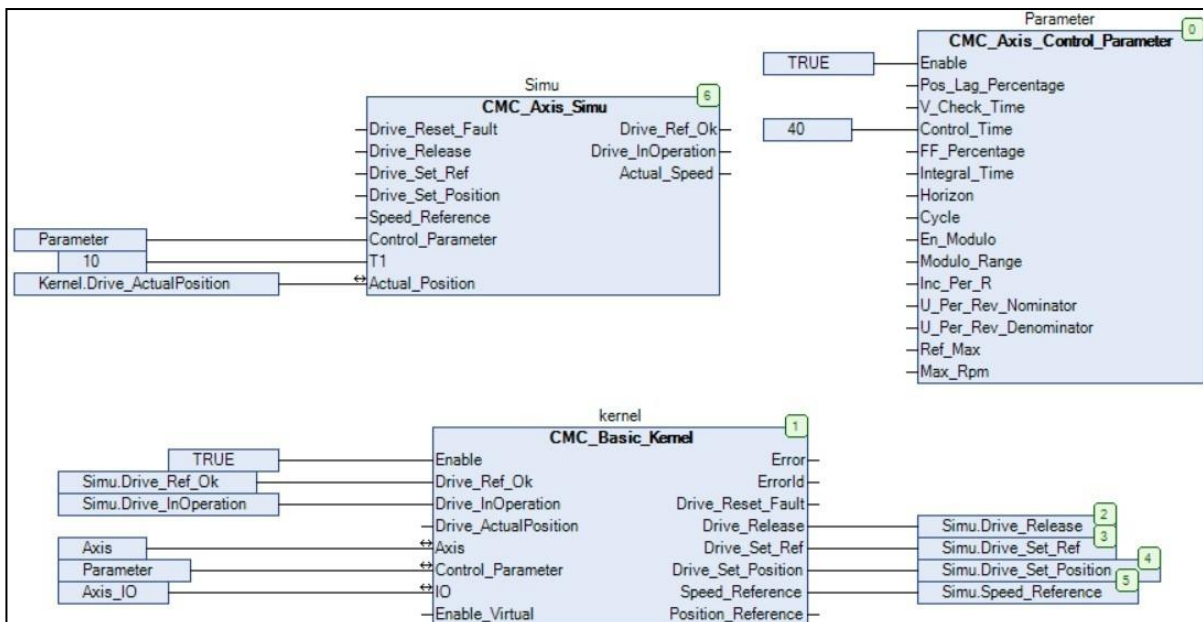
It is possible to use a simulated axis instead of a real drive. The axis simulation can be used in the following use cases:

When the real drive is not available the simulation can be used to test all available motion functionalities to verify the application program.

The simulation can be used to create a virtual master axis and synchronize other axes to it.

The simulation is realized by the Function Block CMC_Axis_Simu or input Enable_Virtual = TRUE can be used at the KERNEL-block.

Homing will be possible if the limit-switches (data type CMC_Axis_IO) are simulated also. This is not done by CMC_Axis_Simu but could be realized in the PLC program.



Example for Simulation

The drive velocity is simulated by PT1-Characteristic. The input T1 gives the time constant for this PT1 as multiple of the cycle time. All other properties are simulated according to the CMC_Axis_Control_Parameter.



Note: The value of the time behavior from the axis simulation Function Block set by the input T1 has to be at least four times smaller than the value of the axis parameter Control_Time from the parameter Function Block. If Enable_Virtual = TRUE is used, no delay will be applied to the simulated drive speed, and it will not be possible to test the position-control loop, but it will be fine to be used as virtual axis.

10.1.4.2.4 How to perform a homing

The homing of an axis is a procedure which consists of up to two phases. For each phase there are different Function Blocks available. The available Function Blocks are according to PLCopen and belong to the application layer. Available Function Blocks for each phase are listed in the table below.

Overview of the available homing function blocks

	Phase 1	Phase 2/Finish Homing
MC_StepAbsSwitch	X	
MC_StepDirect		X
MC_StepLimitSwitch	X	
MC_StepRefPulse		X

To create a complete homing sequence one Function Block of each phase can be used.

First phase

The used Function Blocks will change the axis state to Homing and will move the axis to approach installed limit switches or a dedicated absolute switch in the desired directions. No manipulation of a position value will be done in this phase. The use of Function Blocks of this phase is optional for a homing.

The signals of the installed limit switches have to be written to a variable of the data type CMC_Axis_IO.

Second phase

Function Blocks from this phase will also change the axis state to Homing if this has not already happen and will finish the homing. Therefore a new position will be set to the axis. The axis state will then switch back to Standstill.

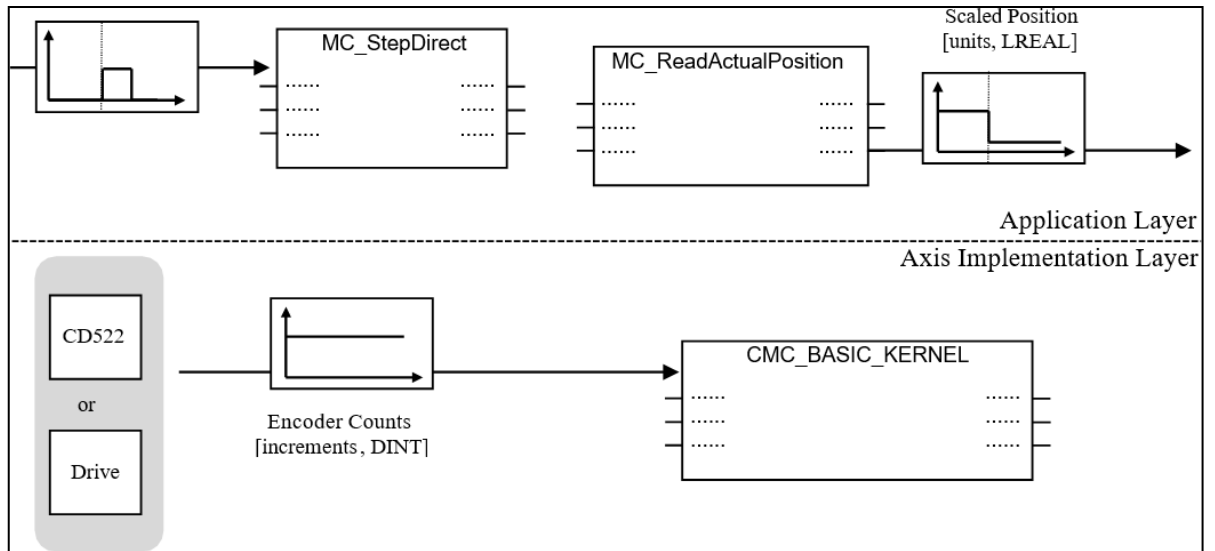
The use of a Function Block of the second phase is mandatory for a homing.

In general with AC500 PLC-based Motion Control there are two position values: One position value will represent the encoder counts of a drive or the CD522 module which is connected to the input Drive_ActualPosition of the kernel Function Block. The other position is a user defined scaled unit which is used for PLCopen Function Blocks.

There are different ways to finish the homing by manipulate and adjust a position value. Which value should be manipulated depends on the used drive or module and its capabilities. See the following types A, B and C.

Type A

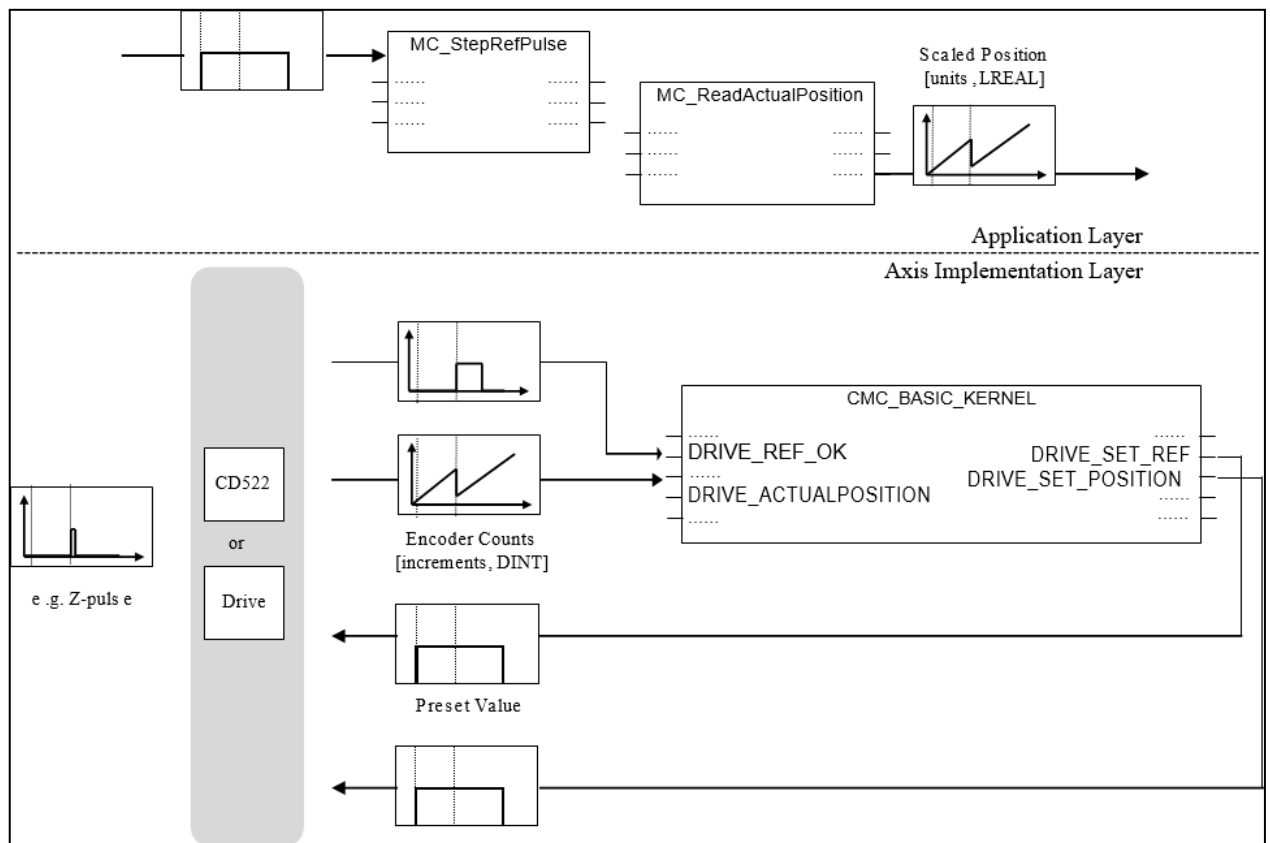
The user defined position unit will be changed only. The Function Block MC_StepDirect must be used here. This type of homing is less complex than the other types but also less precise.



Homing Type A

Type B

The Drive or the CD522 module will change its own position value, the encoder counts.



Homing Type B

The process will be started by the execution of the Function Block **MC_StepRefPulse**. The axis will start to move.

The output **Drive_Set_Ref** of the kernel Function Block will then set the drive to sense for a digital signal. At the same time the kernel Function Block outputs a preset value which will replace the actual encoder count value at the moment the digital signal occurs.

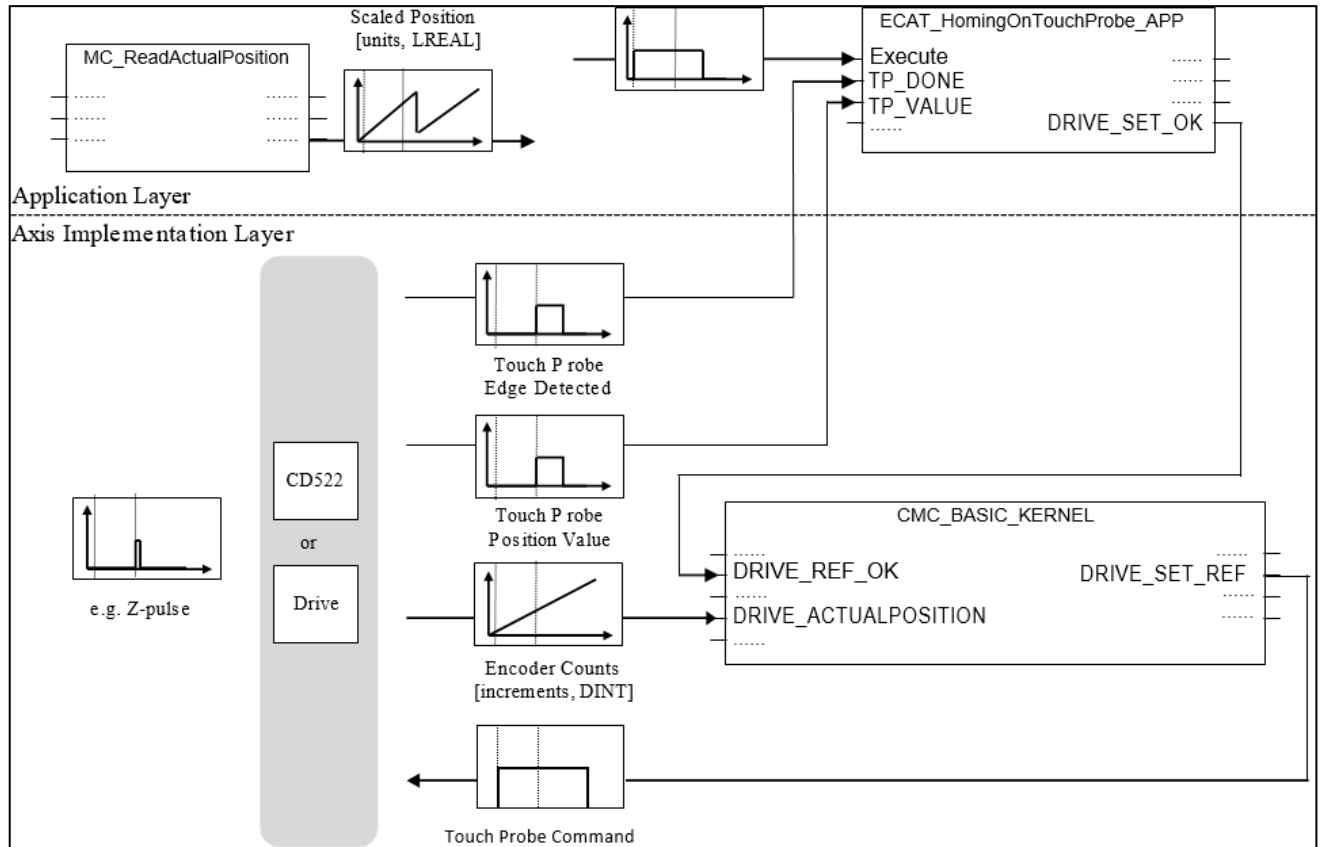
This signal can be a Z-pulse of an incremental encoder but also any other signal from a sensor. This functionality may require a configuration of the drive or the CD522 module to be used.

In the same cycle when the new position value is set there also has to be a Boolean signal stating a new position value at the input Drive_Ref_Ok of the kernel Function Block. The user defined position value will then be shifted accordingly.

Example of type B for phase 2: Chapter “How to connect a drive”

Type C

The encoder count position value will not be changed but involves registration capabilities of a drive or the CD522 module.



Homing Type C

The process will be started by the execution of the Function Block ECAT_HomingOnTouchProbe_APP (ABB_Ecat_CiA402_AC500.library).

The axis will start to move. The output Drive_Set_Ref of the kernel Function Block will then command the drive or the CD522 module to activate the Touch Probe functionality. This will configure the drive to latch a position at the moment a digital signal occurs. The digital signal can be a Z-pulse of an incremental encoder but also any other signal from a sensor. This functionality may require a configuration of the drive or the CD522 module in order to be used.

In combination with the latched position value there is a Boolean signal which states that a new latch value has been received. In case of the module CD522 this encoder count position value has to be converted from encoder counts to equivalent user scaled units by the use of the function “CMC_Get_Units_From_Inc” (ABB_MotionControl_AC500.library) before it can be connected to the Function Block ECAT_HomingOnTouchProbe_APP.

To manage the Touch Probe objects of a drive within the CiA402 profile (e.g. Microflex e190) the Function Block ECAT_HomingOnTouchProbe_APP (ABB_Ecat_CiA402_AC500.library) can be used. This will also cover the conversion from encoder counts to user scaled units.

At the end of the process the Function Block ECAT_HomingOnTouchProbe_APP will manipulate the user scaled position value according to the latched position from the drive and the users settings.

10.1.4.2.5 How to Use a CAM curve

Note – From Automation Builder 2.5.0 onwards the inbuild Cam Editor is the preferred method to generate Cam Table. For more details on how to use Cam Configurator please refer to chapter 9.1.

It is recommended to use the CAM Editor from Automation builder for those who are new to Cam table or to get the structure of the Cam Table. User can create the complete CAM Table using Cam Editor or can make a copy of CAM Table (IEC Code) and adapt it directly in the IEC code if needed.

The below described CAM functionality is only available in combination with the kernel Function Block CMC_Basic_Kernel.

Details on the CAM Table structure and different parameters to be considered while creating the CAM is described below.

General usage

The usage of a CAM function is based on the following elements:

CAM table defined with the data type MC_PProfile.

An instance of the Function Block MC_CamTableSelect

An instance of the Function Block MCA_Cam_Extra (optional)

An instance of Function Block MC_CamIn

An instance of Function Block MC_CamOut

The following steps are necessary to use a CAM table

Declare a CAM table as an array of the data type MC_PProfile in the program and Write data to this array. Usually, this step is done automatically when using Cam editor functionality.

Use the address of the CAM table at the input CamTable of the Function Block MC_CamTableSelect.

Execute the Function Block MC_CamTableSelect to process the data of the CAM table with the Function Block's input parameters

Additionally, you can execute the Function Block MCA_Cam_Extra for optional parameters after the processing of the Function Block MC_CamTableSelect.

Execute the Function Block MC_CamIn to start the slave axis movement according to the CAM table data and parameters.

The axis will operate in the axis state Synchronized Motion.

To leave the axis state you can execute the Function Block MC_CamOut.

The axis state will switch to state Continuous Motion and maintains its last velocity as long as there is no other command.

You can also use any other motion command interrupt the Synchronized Motion.

CAM table

CAM data is done with one table (two dimensional – describing master and slave positions together).

The data of the elements (array of data type MC_PProfile) can either be assigned within the declaration or can be assigned during runtime before the execution of the Function Block MC_CamTableSelect.

It can be filled with data in the following ways:

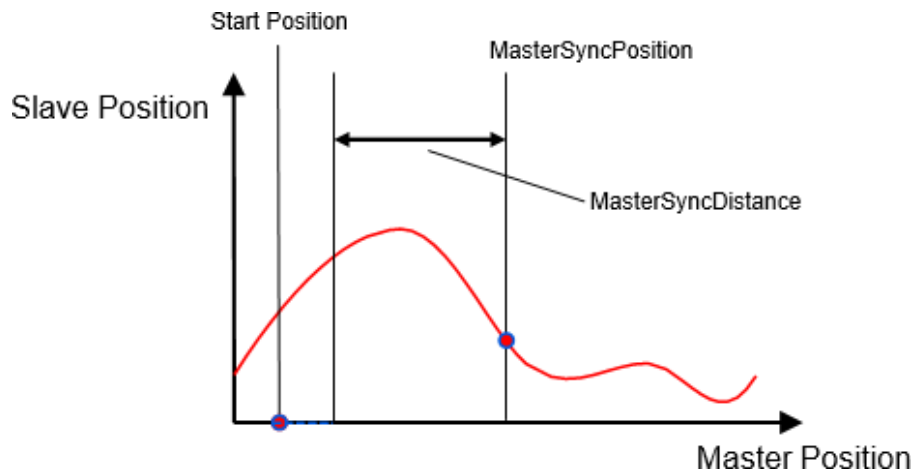
- To use a predefined variable list.
- To calculate the values within the program (before using the MC_CamTableSelect).
- To send values by any communication access to the PLC.

In order to use the new data, it is necessary to execute the Function Block MC_CamTableSelect again. In case the CAM table is executed the Function Block MC_CamTableSelect may not be executed.

The inputs MasterSyncPosition and MasterSyncDistance of the function block MC_CamIn can be used to define a distance to synchronize the slave axis onto the CAM table during the start. In case master axis

moves with negative velocity the parameter MasterSyncDistance can be negative. The MasterSyncPosition should always be within the range of the CAM table master position.

MasterSyncDistance = 0 will deactivate the synchronization. In this case the slave axis should be moved on the CAM curve before MC_CamIn is executed, otherwise a following error can occur.



CAM profile illustration

The master position in the CAM table must be strictly monotonic rising (varying in such a way that it either never decreases).

The length of a CAM table is just restricted by the memory size of the PLC. When long tables are used, it is recommended to call CamTableSelect in a task with lower priority as it will need a considerable computing time.

It is possible to hold several CamTables as a pool and to switch from one to another. This must be done at matching positions as no means for synchronization are available.

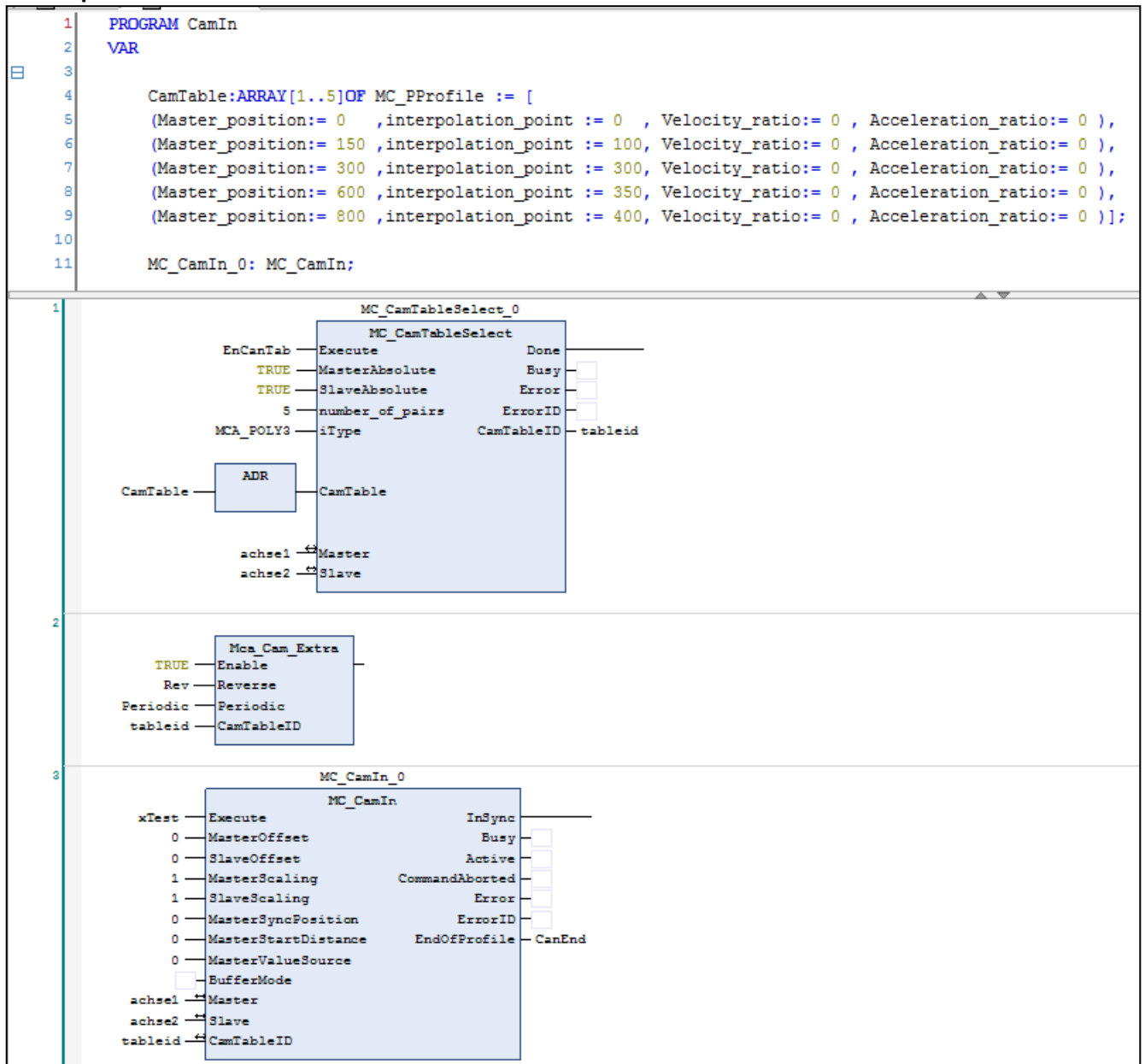
The offset and scaling values (except the time-scale) are transferred continuously. This will allow to follow a "Moving Target" by adjusting these values.

The parameters at MC_CamTableSelect, MC_CamIn and function and MCA_Cam_Extra also modify the behavior:

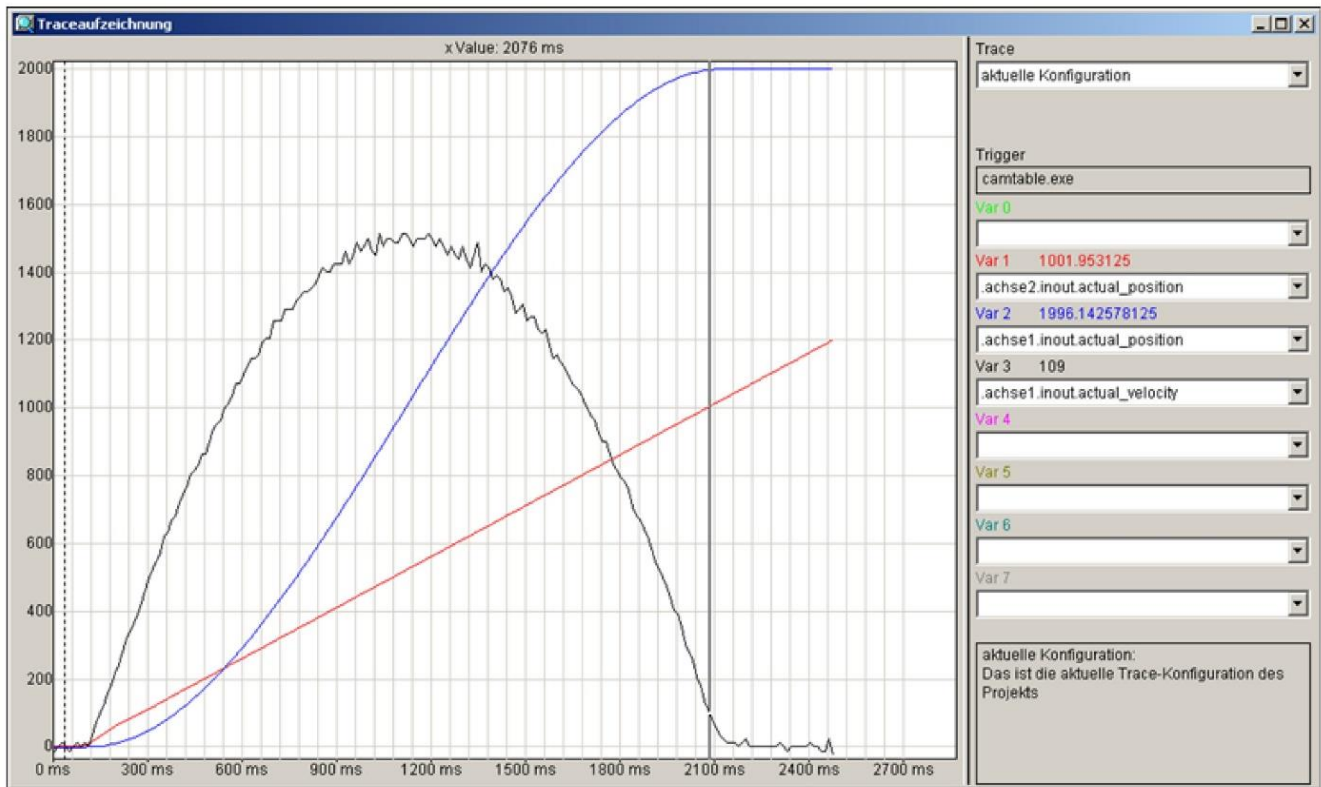
Parameter MC_Cam-TableSelect	Type	Default value	Comment
MasterAbsolute	BOOL	FALSE	TRUE=Master_position from MC_PProfile equals the master axis absolute position. FALSE=CAM is executed relative to the master axis actual position at start.
SlaveAbsolute	BOOL	FALSE	TRUE=interpolation_point from MC_PProfile equals the slave axis absolute position. FALSE=CAM is started from actual slave position. The values "interpolation_point" are relative to the slave axis position at start.
iType	MC_ABB_iTypes_ENUM		Interpolation type.
Number_of_pairs	INT		Number of points used in TimePosition Array.

Parameter MC_CamIn	Type	Defaultvalue	Comment
MasterOffset	LREAL	0	Just used with MasterAbsolute=TRUE, ignored otherwise. Used position for cam-table is: Master axis position-MasterOffset.
SlaveOffset	LREAL	0	Just used with SlaveAbsolute=TRUE, ignored otherwise. Used position is slave axis position=interpolation_point+Slaveoffset.
MasterScaling	LREAL	1	The position used for interpolation is multiplied by MasterScaling, e.g MasterScaling=2, the scaled master will pass the position range with double velocity and within the half distance compared to its real velocity and position.
SlaveScaling	LREAL	1	Interpolation result is multiplied by Slave-Scaling, e.g SlaveScaling=2: Slave axis will run twice the distance.
MasterSyncPosition	LREAL	0	Start synchronization at master axis position = MasterSyncPosition - MasterStartDistance + MasterOffset, meet the CamTable at masteraxis position = MasterSyncPosition. In case of MasterAbsolute = FALSE: start at "actualPosition + MasterSyncPosition - Master- StartDistance", meet the CamTable at "actual-Position+MasterSyncPosition"!!! It is just possible to use the "sync" mechanism when the axis is in StandStill on start.
MasterStartDistance	LREAL	0	A negative value will create a reverse synchronization mode, which means the master should move in negative direction to synchronize. It is independent from the Reverse- Bit which indicates how to end the movement.
These 2 parameters are "extras" to be written with the MCA_Cam_Extra function. When the parameters are used, the MCA_Cam_Extra has to be called after the MC_CamTableSelect.			
Periodic	BOOL	TRUE for master "Modulo", FALSE for master linear axis	CamTable will not reach "EndOfProfile" but will be repeated periodically. When the master is a linear axis, it must move forward and backward within the CamTable position range, but even when it leaves this position range, the CamTable will stay active.
Reverse	BOOL	FALSE	Just necessary when a CamTable is NOT "periodic" and will run in reverse direction (master with negative velocity) Reverse=FALSE, the CamTable is ready when the master leaves the position range in positive direction, e.g. when it moves from 359° to 0° on a rollover axes Reverse=TRUE, the CamTable is ready when the master leaves the position range in negative direction.

Example for CAM curve

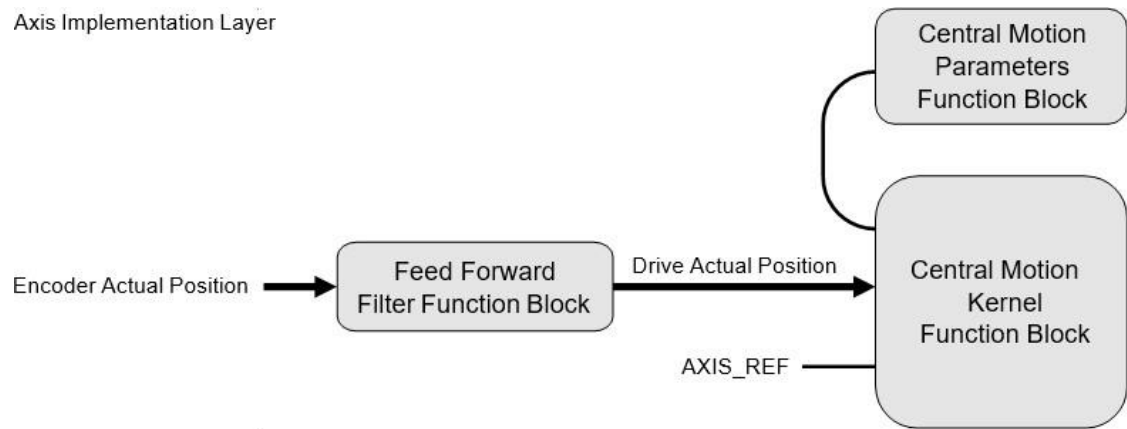


In the example, the slave will run from 0 to 2000 while the master runs from 0 to 1000. The slave will start and end with velocity=0, no matter which velocity the master has during start. The slave will reach the maximum velocity when it is at position 1000 and the master is at position 500.



10.1.4.2.6 How to use an external axis

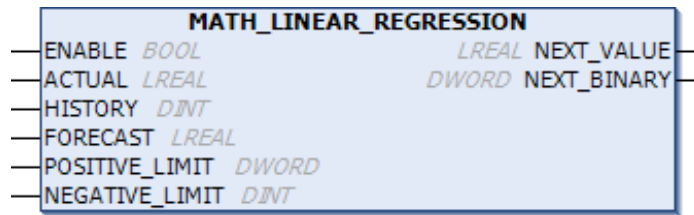
To use multiaxis PLCopen Function Blocks with an externally sensed axis as master axis the following structure can be used for the axis implementation:



Structure synchronization to an external axis

The use of a feed forward filter Function Block is needed if the slave axis has to follow the position of the external axis. In this case there will be a time delay between sensing the position of the external axis and moving the follower axis along the sensed position. The filter Function Block will then add a certain distance to the external axis' position depending of its speed.

The filter Function Block MATH_LINEAR_REGRESSION from the library ABB_MathFunctions_AC500.library can be used here.



For an axis which is following the external axis, the value “mcActualValue” (from MC_Source enumeration) for the input “MasterValueSource” for multi-axis PLCopen Function Blocks has to be used.

When the filter Function Block MATH_LINEAR_REGRESSION is used to process an actualposition, 2 different purposes are fulfilled:

A jitter or noise can be compensated

It is possible to calculate a forecast-position to compensate for a delay in position measurement



Note: Process the actual position or any other master axis always before the slave axis. Otherwise, an additional one cycle-delay is introduced.

The MATH_LINEAR_REGRESSION function block calculates the progress for a variable which is captured in equidistant periods of time and is assumed to follow a linear curve. It uses the Gauss “least squares” - algorithm to do so. The line is calculated in a way that the sum of squares for the distances from the measured points to the assumed straight line is minimized.

A noise or jitter influence of the value is compensated and a predictive value for the variable with an adjustable forecast horizon can be calculated.

Linear equation:

$$\text{Line}[i] = \text{gradient} * i + \text{offset}$$

Sum of squares:

$$\text{sum} = \sum_{i=1}^{\infty \text{history}} (x[i] - \text{line}[i])^2$$

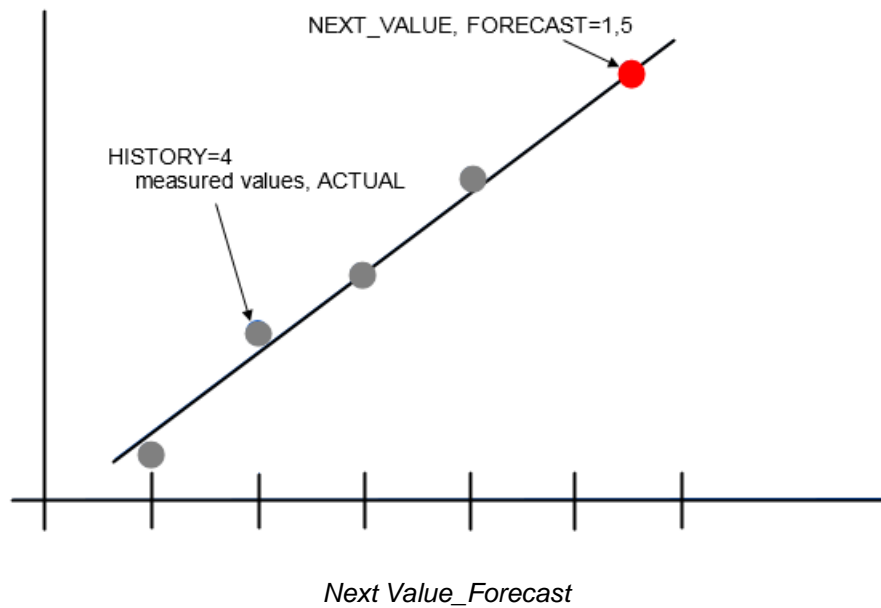
The gradient and offset for the line are calculated in a way that “sum” is minimized. Then these 2 values are used to calculate the forecast value:

$$\text{NEXT_VALUE} := \text{gradient} * \text{FORECAST} + \text{offset}$$

FORECAST=0 would mean: value right now, no future or past considered.

When the ACTUAL value is a modulo value, for example a single turn encoder or a rollover axis, this has to be considered in the calculation. The 2 input values POSITIVE_LIMIT and NEGATIVE_LIMIT can be used to configure this. They define the upper and lower limit for ACTUAL. Also, the NEXT_BINARY will as a result be limited to these borders.

Example



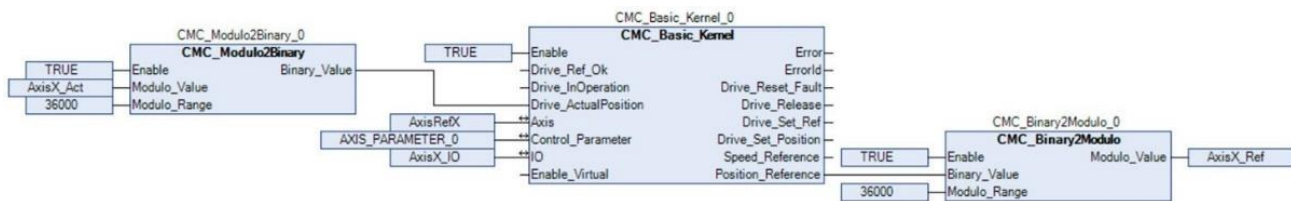
10.1.4.2.7 How to use an encoder/drive with <> 32 bit position overrun

The incremental position as actual position at the Function Block CMC_Basic_Kernel is usually assumed as position with a 32-bit position overrun. As well as it is the reference position which is sent to the drive.

Any modulo-axis configuration should be done inside the PLC.

Some drives are requested to correct their positions themselves for a non-linear axis which should constantly run into the same direction.

In this case, the drive has to be configured as a modulo-axis and the Function Block CMC_Basic_Kernel needs some additional Function Blocks to create the 32-bit value Chapter “Roll-Over axis”



Kernel

The Function Block CMC_Modulo2Binary will convert any position with any Modulo_Range to a 32-bit binary position.

The actual_position is assumed to run between 0 to Modulo_Range.

The actual_position should not change > 1/4 Modulo_Range between two scan cycles.

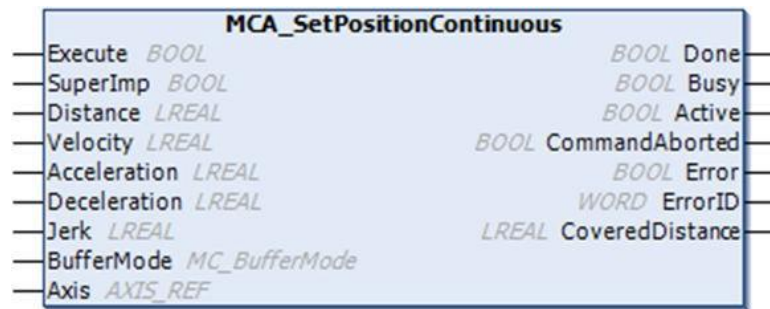
The Function Block CMC_Modulo2Binary will convert the 32-bit binary position reference from CMC_Basic_Kernel to a position reference which runs from 0 to Modulo_Range.

10.1.4.2.8 How to do position correction "on the fly"

Sometimes it is required to have a position correction "on the fly". For example, it can happen that a position is wrong due to mechanical slip and that a switch which is passed by during the movement is used to capture a position value.

In other cases, it is required to synchronize the position to a print mark, so an actual_position has to be corrected, but not the movement of the printed material.

For both applications, the Function Block `MCA_SetPositionContinuous` can be used. It will use ramps and a limited velocity for the correction, so it will be tolerable to execute it during an ongoing movement and while the axis is activated in a multi-axis movement.



MCA_Set_PositionContinuous_V3

The block can be used in any axis state except `ERRORSTOP` and `HOMING`. Two different operation modes are possible:

1. SuperImp=FALSE

The actual_position will be modified.

The block will not cause any movement.

If a `PLCopen` block in `DISCRETE_MOTION` (positioning) is active during the execution, this block will not reach Done as the actual_position is modified.

If a slave axis is coupled to an axis while `MCA_SetPositionContinuous` is executed (with SuperImp=FALSE) it will follow.

This mode is possible while the axis is in state `DISABLED`.

2. SuperImp=TRUE

The actual_position will stay constant.

A mechanical movement is executed (without changing the axis state machine).

A slave axis will not follow.

This behavior is like a superimposed movement.

It is not possible when the axis is in state `DISABLED`.

The block can just be aborted by another `MCA_SetPositionContinuous`.

10.1.4.2.9 How to limit the movement

It is possible to limit the movement by position (software limit switches) and by velocity. By default, no software limit switches are activated in `PS5611-Motion`. It is possible to activate them by accessing some `PLCopen` parameter.

The functionality described below is just available with linear axes.

Num-ber	Parameter	Data type	Minimum	Maximum	Default	R/W	Descrip-tion
2	SWLimitPos	DINT	2147483647	2147483647	2147483647	R/W	Positive Softwarelimit switch position.
3	SWLimitNeg	DINT	2147483647	2147483647	2147483647	R/W	NegativeSoftwarelimit switch position.
4	EnableLimit-Pos	BOOL	FALSE	TRUE	FALSE	R/W	Enable positive softwarelimit switch.
5	EnableLimit-Neg	BOOL	FALSE	TRUE	FALSE	R/W	Enable negativesoftwarelimit switch.
2003	Enable-Limit2Decelerate	BOOL	FALSE	TRUE	FALSE	R/W	Enable softwarelimit switchesto decelerate
2004	EnableLimit-Abort	BOOL	FALSE	TRUE	FALSE	R/W	Enable that soft-ware limit switches will abort ongoing movement FALSE = Limits position and velocity, decelerates and shows awarning until the position limit is reached,then ERROR STOP TRUE = Switches off any ongoing motion and decelerates to the position limit, then ERROR STOP
2005	Enable-LimtVelocity	BOOL	FALSE	TRUE	FALSE	R/W	If the velocity islimited theunmoved position will be covered wheneverpossible
2006	SWLimit2DecPos	LREAL	-2147483647	2147483647	2147483647	R/W	Used as end position for Enable- Limit2Decelerate
2007	SWLimit2DecNeg	LREAL	-2147483647	2147483647	2147483647	R/W	Used as end position for Enable- Limit2Decelerate
2008	MaxPositioGap	LREAL	0	214748364700	0	R/W	Used to stop the ongoing movementif position is behind

The following different behavior is possible:

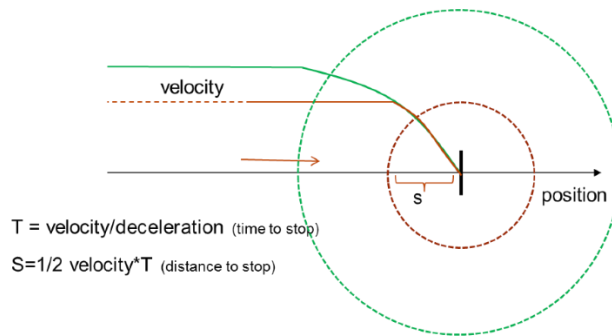
- No limitation at all (default)
- Limit position with ERRORSTOP: Limit position between SWLimitNeg to SWLimitPos, axis to state ERRORSTOP in case the position range is left.
- Limit velocity and acceleration: Limit velocity to paraMaxVelocityAppl and acceleration/deceleration to paraMaxDecelerationAppl, create WARNING_VELOCITY, not state changes for axis, abort movement is optional when MaxPositionGap is reached due to limitation.
- Limit Position with ramp-down: In addition, it is possible to limit the position between SWLimit2DecNeg and SWLimit2DecPos. paraMaxDecelerationAppl is used to ramp down.
- When activated with EnableLimitPos or EnableLimitNeg, the reaction will be as follows: When the control position reaches the respective limit switch, the axis will go to state ERRORSTOP, and Drive_Release will

be switched off. The actual_position might be behind, depending on the following error. It is assumed that a drive or application specific braking is performed. The axis will be stopped behind the limit.

The axis could be switched on again by MC_Power. A movement in the opposite direction will be possible.

The functionality of EnableLimitPos and EnableLimitNeg is unchanged.

You can use the limitation of movement to achieve a soft or adjustable braking in advance before reaching the software limit switch. The limitation is activated by three Boolean parameter and will calculate a position distance to the limit switch, which depends on the actual velocity and given deceleration ramp. "paraMaxDecelerationAppl" is used for deceleration. It will decelerate the axis by the given deceleration ramp when the calculated position is reached and stop at the software limit switch. The original behavior is not modified, so if also these software limit- switches are activated, the axis might be set to state ERRORSTOP.



There are 2 different modes:

EnableLimitAbort = TRUE

Any ongoing motion will be aborted immediately (when the distance to stop is reached, as shown in the above diagram), a warning is shown

The axis will be decelerated to reach the software limit switch.

EnableLimitAbort = FALSE, EnableLimitDecelerate = TRUE

A warning is shown and the velocity is reduced, with respect to the given deceleration and position limit.

The ongoing motion is not aborted. If it was just a "tight fit", e.g. in a master slave movement and the direction is turned soon enough, it might be possible to continue the movement.

As the ongoing movement is not interrupted, an activated movement might not be completed, for example a MC_MoveAbsolute will never reach its target position. A warning is shown at function block CMC_Basic_Kernel.

When EnableLimitPos = TRUE or EnableLimitNeg = TRUE, and the values for SWLimitPos or SWLimitNeg are set, the axis will be set to state ERRORSTOP when these position limits are reached.

In addition, the function block will allow to limit the velocity. With EnableLimitVelocity = TRUE, it will monitor the velocity demand from the position reference and limit the position reference, so the given velocity limit will not be exceeded. A warning will be shown. The velocity used for limitation is MaxVelocityAppl.



Note: The velocity limitation can be used to prevent short-term velocity peaks. The limited position will be caught up later, whenever possible. This can result in not-expected behavior. The WARNING issued by CMC_Basic_Kernel can be checked and used to stop a movement. The movement will be aborted automatically when the position is by MaxPositionGap behind.

For a single axis movement, the commanded velocity is limited at the beginning. No position gap will occur.

In a multi-axis movement, the slave axis follows a master. This can result in a position gap. A velocity peek from the master axis can be reduced by using the limitation. If the master is too fast because of the value for MaxPositionGap, the movement will be aborted

When EnableLimit2Decelerate or EnableLimitAbort are used, the velocity is limited to MaxVelocitySystem with EnableLimitVelocity = FALSE. The function modifies the position reference. This modified position reference is used to control the drive. Whenever the limitation interferes the kernel will show a warning or an error. The warning or error message will disappear when the situation is cleared.

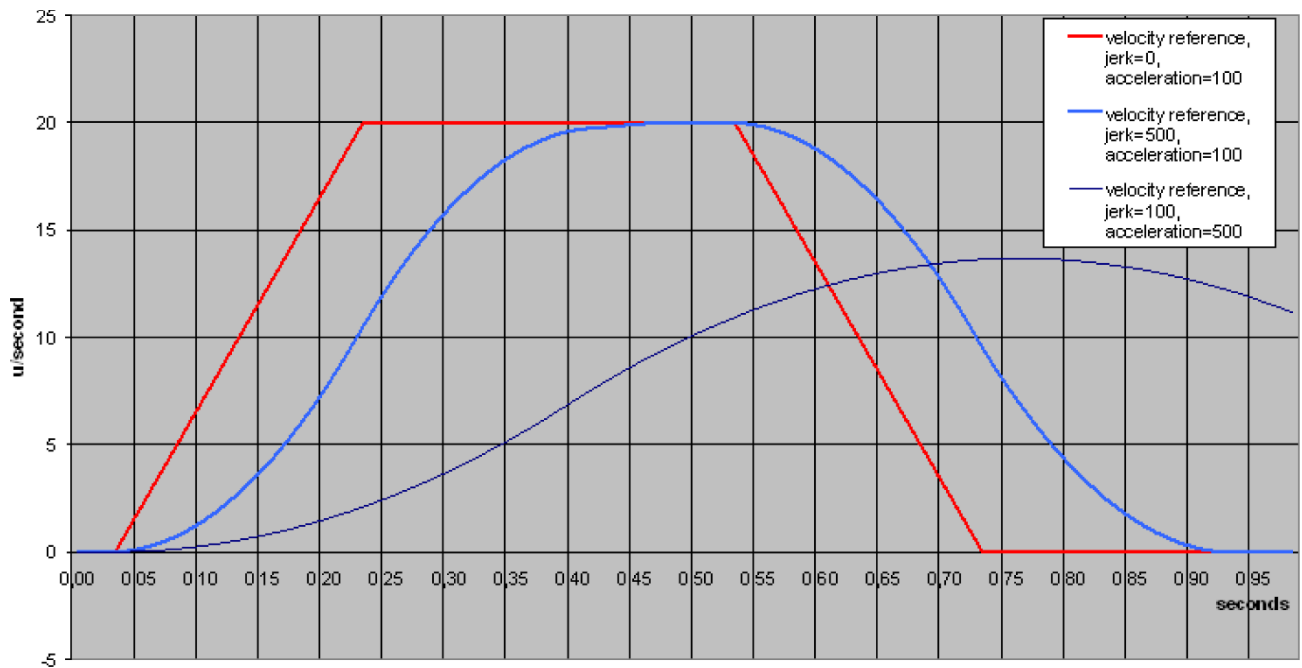
Parameter Number	Parameter Name	Value	Comments
4	EnableLimitPos	TRUE	ERRORSTOP when positionsexceed, no previous warning or deceleration.
5	EnableLimitNeg	TRUE	
2003	EnableLimit2Decelerate	FALSE	
2004	EnableLimitAbort	FALSE	
2005	EnableLimtVelocity	FALSE	

Parameter Number	Parameter Name	Value	Comments
4	EnableLimitPos	FALSE/TRUE	Reduce the velocity when reaching a position limit within the deceleration distance calculated by using MaxDecelerationAppl. Display a warning at CMC_Basic_Kernel. The underlying movement stays active. With EnableLimitPos = TRUE or EnableLimitNeg = TRUE: When the Position limit is reached, the axis is set to mode ERRORSTOP also if EnableLimitPos or EnableLimitNeg are used. Otherwise, just the movement is limited, without affecting the state machine. An activated positioning movement will not reach its target. Velocity is limited to MaxVelocitySystem.
5	EnableLimitNeg	FALSE/TRUE	
2003	EnableLimit2Decelerate	TRUE	
2004	EnableLimitAbort	FALSE	
2005	EnableLimtVelocity	FALSE	

Parameter Number	Parameter Name	Value	Comments
4	EnableLimitPos	FALSE/TRUE	Reduce the velocity when reaching a position limit within the deceleration distance calculated by using MaxDecelerationAppl. Display a warning at CMC_Basic_Kernel. The underlying movement stays active. With EnableLimitPos = TRUE or EnableLimitNeg = TRUE: When the Position limit is reached, the axis is set to mode ERRORSTOP also if EnableLimitPos or EnableLimitNeg are used. Otherwise, just the movement is limited, without affecting the state machine. An activated positioning movement will not reach its target. Velocity is limited to MaxVelocitySystem. The active PLCopen function block is aborted as soon as the warning is issued. With EnableLimitPos = TRUE or EnableLimitNeg = TRUE: When the Position limit is reached, the axis is set to mode ERRORSTOP.
5	EnableLimitNeg	FALSE/TRUE	
2003	EnableLimit2Decelerate	---	
2004	EnableLimitAbort	TRUE	
2005	EnableLimtVelocity	FALSE	

Parameter Number	Parameter Name	Value	Comments
4	EnableLimitPos	---	The velocity is checked and also limited to the value Max-VelocityAppl. A warning is shown. The active movement is not aborted. This functionality works independent from software limit switches.
5	EnableLimitNeg	---	
2003	EnableLimitDecelerate	---	
2004	EnableLimitAbort	---	
2005	EnableLimitVelocity	TRUE	

10.1.4.2.10 How does the parameter for jerk influence the axis movements?



Velocity reference with different jerk values

The diagram shows the result with different jerk values and the same velocity and acceleration. The time needed for acceleration with $\text{jerk}=0$ is:

$\text{Time1} = \text{velocity} / \text{acceleration} = (20 / 100) \text{s} = 0.2 \text{s}$ The additional time with $\text{jerk}=500$ will be: $\text{Time2} = \text{acceleration} / \text{jerk} = (500 / 100) \text{s} = 5 \text{s}$ So the total time is:

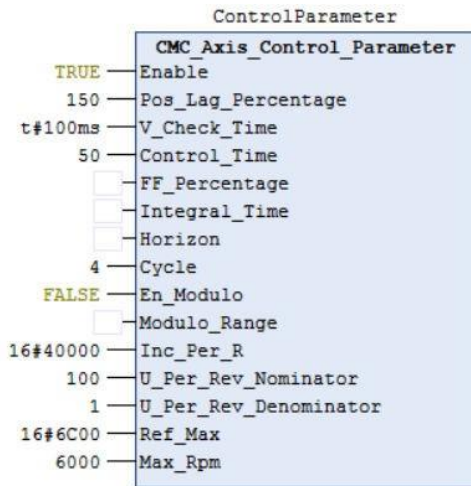
$\text{Time} = \text{Time1} + \text{Time2} = 0.2 \text{s} + 5 \text{s} = 5.2 \text{s}$

In the last example with $\text{jerk}=100$, the velocity and acceleration values are not reached.

10.1.4.3 Axis parameters

The parameters for axis configuration and adjustment are set by the Function Blocks `CMC_Axis_Control_Parameter`.

Depending on the version of the kernel Function Block the corresponding version of the parameters Function Block must be used. The instance will then be connected to the kernel Function Block by its instance name.



In this example the control structure is a simple position control loop with just proportional gain. When the application does not require minimized position following error it should be used this way as it is simple to adjust, robust and requires minimal performance. The proportional gain is then adjusted by `Control_Time`. Just change values at `CMC_Axis_Control_Parameter` when the position control loop is open (`Drive_Release=FALSE`, the axis state is Disabled). The values are sending to the control loop with a positive edge at "Enable". The `CMC_Basic_Kernel` function block needs to be already enabled.

10.1.4.3.1 Supervision

Pos_Lag_Percentage

This parameter configures the position window for the supervision of the following error. The default value is 150[%]. A value of 0[%] will deactivate the supervision function.

The size of the position window depends on the setting of the parameters `Control_Time` and `Max_Rpm` "Control_Time".

$$\text{Position Window [Increments]} = (\text{Inc_Per_R}) * (\text{Max_Rpm}/60) * (\text{Control_Time}/1000)$$

$$\text{Position Window [Units]} = (\text{U_Per_Rev_Nominator} / \text{U_Per_Rev_Denominator}) * (\text{Max_Rpm}/60) * (\text{Control_Time}/1000)$$

Example

$$\text{Position Window [Increments]} = (10000) * (6000/60) * (50/1000) = 50000 \text{ [Increments]}$$

$$\text{Position Window [Units]} = (1/1) * (6000/60) * (50/1000) = 5 \text{ [Units]}$$

A value of 100% will result in a position window which corresponds to the expected following error with the giving `Control_Time` at `Max_Rpm`. Therefore it is recommended to use values higher than 100[%]. In case the parameter `FF_Percentage` is used smaller values can be used.

If the supervised position window is exceeded the axis state will change to `ERRORSTOP`.

V_Check_Time

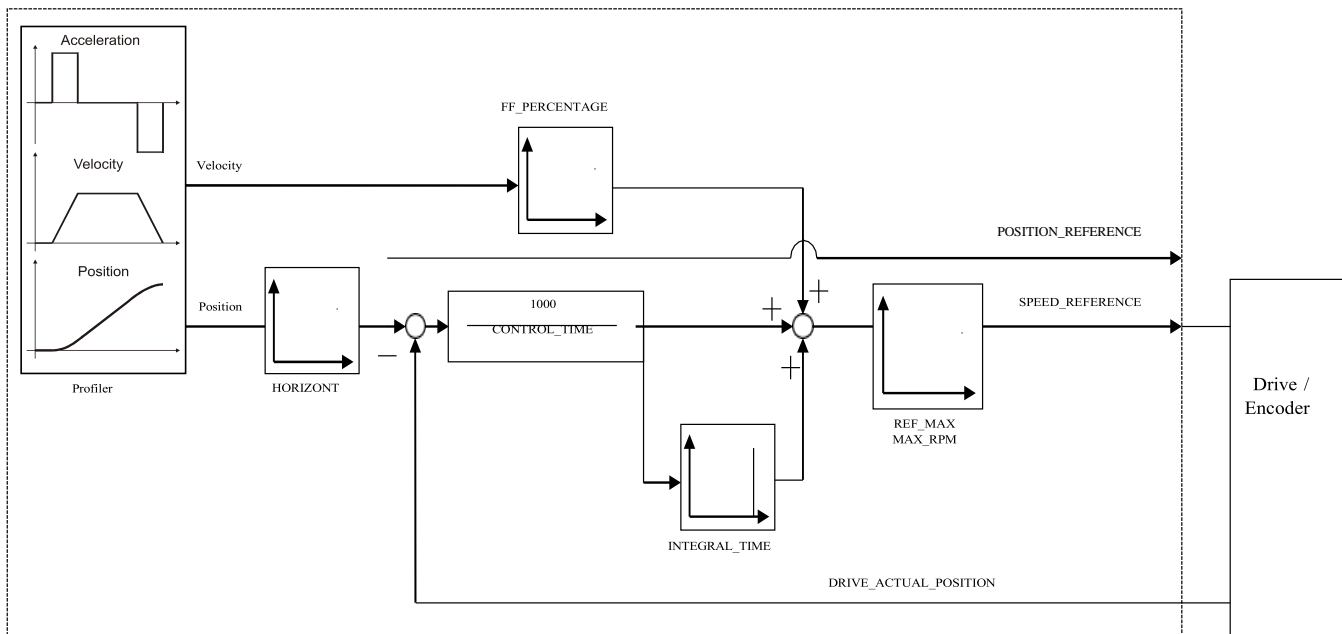
After the configured time, the drive's actual velocity must be at least 50 % of the commanded velocity. This function can also be used in case the Position Reference is transferred to the drive.

A value of 0 will deactivate this supervision function.

If the supervised velocity window is exceeded the axis state will change to `ERRORSTOP`.

10.1.4.3.2 Position control loop



Kernel Function Block

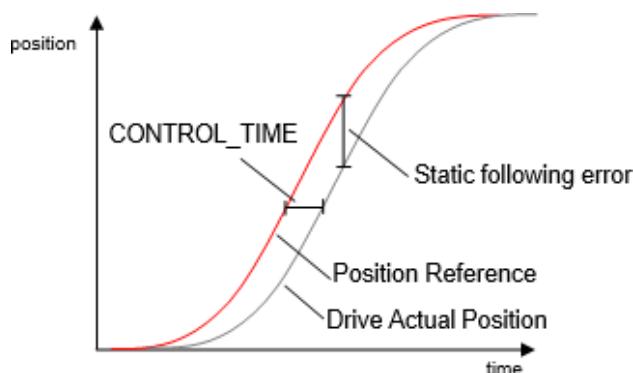


Basic structure of position control loop

Control_Time

The default value is 100 which leads to a proportional gain of 10.

	Note: In case the value of Control Time is too short the position control loop will run into instability.
	Note: In case the position control loop has not used this parameter must not be set to 0.

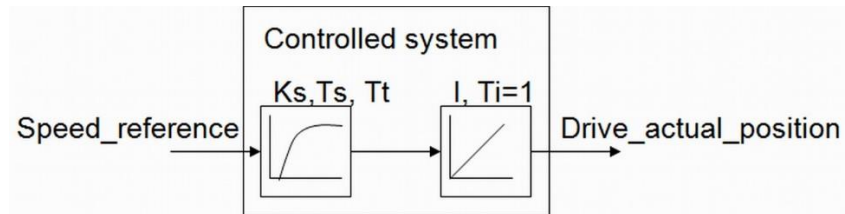


Control Time and static following error in case the feed forward of velocity and the inte-grational part of the position control loop is not used.

The static following error depends on the axis velocity and can be calculated easily: $p_error = v * CT$.

In general it should be aimed to reach a *high* position control loop gain with a *short* Control Time to achieve a small following error. As the reaction times take account in the possible Control Time of the complete system (parameters of the drive control loop, PLC cycle time as well as the communication fieldbus) should be considered.

As a basic rule the Control Time should be at least four times longer than the reaction time between the output of the Speed Reference and the input of actual position.



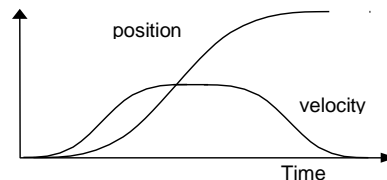
When the time T_s and T_t is measured, a control_time of $4 * (T_s + T_t)$ will result in an aperiodic damping of the position control loop. It is important to measure the values from inside the PLC (e.g. Trace) to have the complete reaction times included. Practical values for Control_Time might be from 50 - 500ms. The PLC cycle time as well as bus cycle times and mechanical reaction will influence the value.

FF_Percentage

The default value is 0.

In case a velocity feedforward must be configured a value of up to 80 is recommended. For larger values than 80 the parameter Horizon needs to be used as the resulted position will over- shoot otherwise.

A value of 100 adds a velocity to the Speed Reference output which corresponds exactly to the ongoing Position Reference value.



Integral_Part

The integral part of the position control loop can be used to eliminate a permanent positioning error, e.g. in case of hanging loads.

The time value can be regarded as the time the integrator needs to sum up the input value to reach the same value for its output.



Note: In case the Integral Part Time is too short the position control loop will run into instability.

Horizon

A communication delay of the Speed Reference value to the drive system can cause an over-shoot during positioning caused by the velocity feedforward gain.

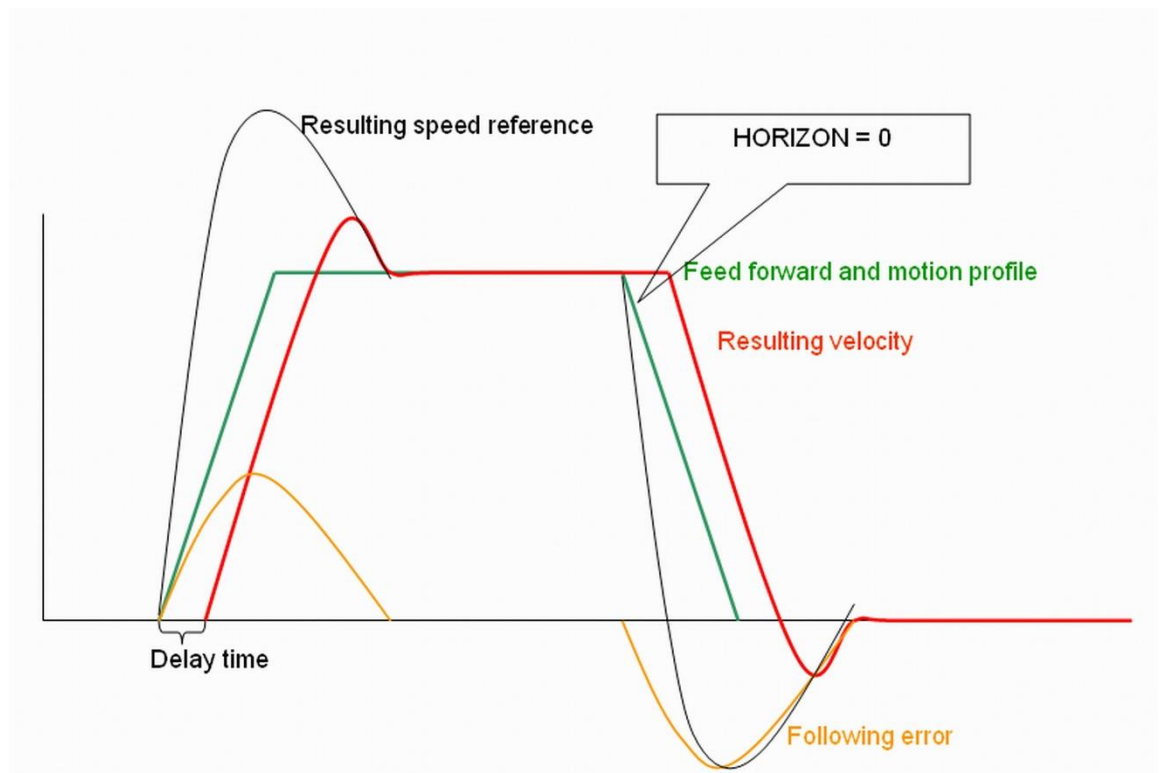
This function will compensate this communication delay to prevent an overshoot by time shifting the signals Velocity Feed Forward and Position Reference relatively to each other.

The value of Horizon can be assumed to be approximately the time delay of the communication delay.

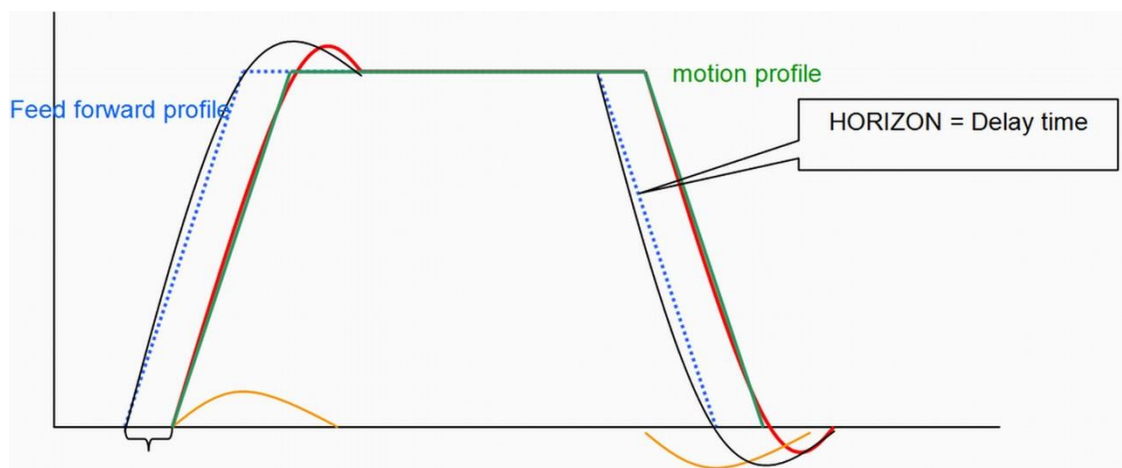
The delay time might be caused by the cycle time of the control loop and by any delay in sending the speed reference, delay in the drive to build up the torque and delay to receive the actual position. To overcome this delay, a Horizon > 0 might be used. The feed forward reference will be created in advance, while the proportional gain is applied to the original motion profile. The delay is then compensated.

This function should not be used if the feed forward parameter FF_Percentage is 0. A value of 0 will deactivate this function, which is the default value.

While this function is used, it will increase the needed PLC calculation time for this axis.



Result with Horizon=0



Result with Horizon>0

10.1.4.3.3 PLC Cycle time

This parameter represents the cycle time in which the kernel Function Block of the axis is called. If the configured cycle time is not correct the resulting acceleration and speed of an axis will be not correct also.

In case the task execution of the axis is synchronized to a fieldbus (e.g. EtherCAT) the cycle time of the fieldbus has to be used

10.1.4.3.4 Roll-Over axis

If the Position Reference value is used, the drive must able to perform a position over-run after 32 bit. If the drive's position over-run is different, it can be adapted with the

function blocks CMC_Binary2Modulo and CMC_Modulo2Binary from the library ABB_MotionControl_AC500.library. Incompatibility can cause an axis to trip after hours of operation.

The possible position following error must be smaller the $\frac{1}{2}$ Modulo_Range. Make sure that the modulo range is large enough.

Position following error = $(100 - \text{FF_Percentage}) * \text{Max_Rpm} * \text{Inc_Per_R} * \text{Control_Time} / 6000000$. This is the maximum value at constant velocity.

En_Modulo

With this parameter the axis can be configured as a roll-over axis.

Modulo_Range

The modulo range will be defined in drive position counts (DINT). It will result that the scaled unit position which is used by the PLCopen function blocks will stay within the defined range.

Example

```
En_Modulo           := TRUE
Modulo_Range        := 20000
Inc_Per_Rev         := 10000
U_Per_Rev_Nominator := 360 (example – degree)
U_Per_Rev_Denominator := 1
```

The scaled unit's position will cover the range from 0 to 720 (degrees)

In some cases it is not suitable to set the modulo range of an application with the DINT value of the parameter Modulo_Range only. In such cases the parameters 2001 Modulo_Nominator and 2002 Modulo_Denominator can be used to scale the parameter Modulo_Range to a more precise value.

Parameter Modulo_Nominator and Modulo_Denominator (supported with CMC_Basic_Kernel)

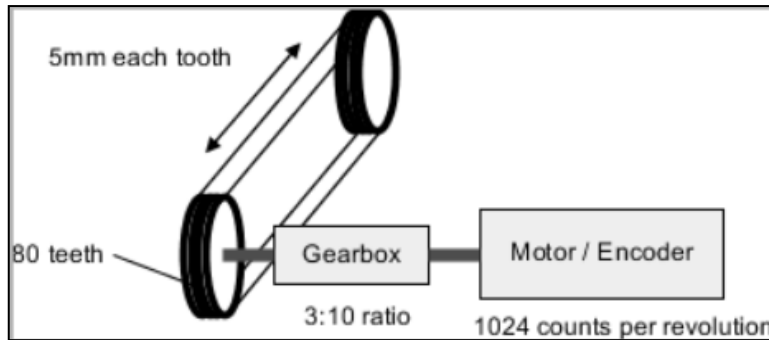
These parameters can be used to modify the Modulo_Range in a way that fractions of an increment could be used for 1 modulo (=rollover) distance

- Default: Modulo_Nominator=1 and Modulo_Denominator=1: the actual position for an axis is limited between 0 and Modulo_Range increments.
- Limitations: $\text{Modulo_Range} * \text{Modulo_Nominator} < 2147483647$. Otherwise: default values will be used.
- When modifying these parameters, the position control loop should be opened.

Example

```
En_Modulo           = TRUE
Modulo_Range        = 1024
Modulo_Nominator    = 10
Modulo_Denominator  = 3
Inc_Per_R           = 1024
U_Per_Rev_Nominator = 80*5*3
```

U_Per_Rev_Denominator = 10



Result of parameters Modulo_Range, Modulo_Nominator and Modulo_Denominator: The modulo range will cover one revolution of the toothed belt wheel.

Result of parameters U_Per_Rev_Nominator and U_Per_Rev_Denominator: One scaled unit corresponds to one mm of the tooth belt.

Example: Gearbox 10.1		
	Option1	Option2
En_Modulo	TRUE	TRUE
Modulo_Range	10240	10240
Modulo_Nominator	1	1
Modulo_Denominator	1	1
Inc_Per_R	1024	10240
U_Per_Rev_Nominator	36	360
U_Per_Rev_Denominator	1	1
Max_Rpm	3000	300

The two options above describe exactly the same configuration. The Modulo_Range is equivalent to 10 motor revolutions and is 10240 increments. For the position, 1u means 1° and the resolution is $360^\circ/10240\text{inc} = 0,035^\circ/\text{Inc} = 1^\circ/28,44 \text{ Inc}$.

Example: Gearbox 10.3		
	Option1	Option2
En_Modulo	TRUE	TRUE
Modulo_Range	1024	10240
Modulo_Nominator	10	1
Modulo_Denominator	3	3
Inc_Per_R	1024	10240
U_Per_Rev_Nominator	108	1080
U_Per_Rev_Denominator	1	1
Max_Rpm	3000	300

The two options above describe exactly the same configuration. The gearbox is 10:3, so the Modulo_Range is equivalent to $1024 \cdot 10/3 = 3413 + 1/3$ increments. For the first option, the resulting modulo range is calculated $1024 \cdot 10/3$, for option2, it is $10240 \cdot 1/3$. For the position, 1u means 1° and the resolution is $108^\circ/1024\text{inc} = 0,105^\circ/\text{Inc} = 1^\circ/9,481 \text{ Inc}$.

10.1.4.3.5 Scaling of the unit of length

Inc_Per_R

With this parameter the number of the drive position counts each revolution of the motor (DINT) have to be entered.

U_Per_Rev_Denominator & U_Per_Rev_Nominator

With these two parameters the number of units which correspond to one revolution of the motor have to be entered.

The units of length can be scaled to values like: mm, inch, degree, ...

All dynamic parameters of the PLCopen function blocks like velocity, acceleration and jerk are based on seconds. Velocity [units/s], acceleration [units/s²], jerk [units/s³]

Example 1

Inc_Per_Rev = 10000

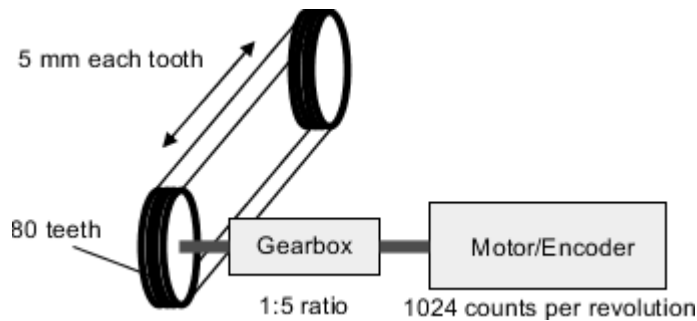
U_Per_Rev_Nominator = 360

U_Per_Rev_Denominator = 1

This will scale one unit to one degrees of the motor shaft. Correspondingly a velocity [units/s] of 360 will turn the motor shaft one revolution per second.

Example 2

In the example one unit will be scaled to one millimeter of the conveyor.



How many units will pass after one revolution of the motor? $(80 \times 5\text{mm}) / 5 = 80$

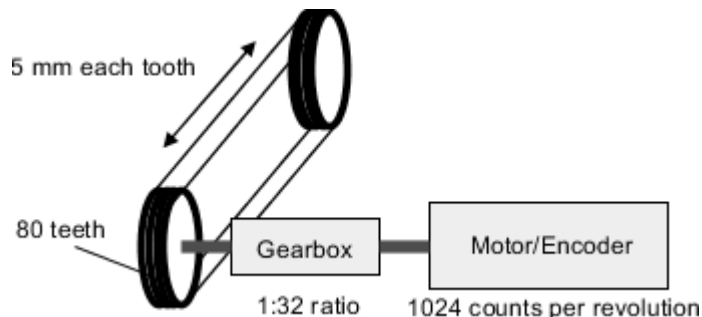
Inc_Per_Rev = 1024

U_Per_Rev_Nominator = 80

U_Per_Rev_Denominator = 1

Example 3

In the example one unit will be scaled to one millimeter of the conveyor.



How many units will pass after one revolution of motor? $(80 \times 5\text{mm}) / 32 = 12,5 = 125 / 10$

Inc_Per_Rev = 1024

U_Per_Rev_Nominator = 125

U_Per_Rev_Denominator = 10

10.1.4.3.6 Scaling of the speed reference output

These two parameters are used to scale Speed Reference output of the kernel FB in order to reach the intended velocity by the output value and to limit the highest possible output value.

Ref_Max

Highest possible output value of the Speed Reference output. The Speed Reference value that corresponds to the parameter Max_Rpm should be used.

Max_Rpm

Maximum speed of the motor in revolutions per minute.

Example

Analog Drive: 1000 rpm at 2 Volts, 3200 rpm at 6,4 Volts (max.)

Analog output module: 10 Volts output at digital value 27648

$\text{Ref_Max} = 17695 (= 27648 / 10 * 6,4)$

$\text{Max_Rpm} = 3200$

10.1.4.3.7 Access and modify parameters



Note: All modifications will be effective immediately. There is no extra plausibility check and values are not checked for limitations.

Use this functionality with care.

Some parameters are collected inside a structure in Axis_Ref, and can be accessed and modified immediately. They are the same parameters as used with function blocks MC_WriteParameter and MC_ReadParameter ♦ Chapter “PLCopen parameter” .

The differences are:

- Only available with CMC_Basic_Kernel
- The parameter values are LREAL instead of DINT and can be used with decimals.
- The parameters will be effective immediately.
- There is no check for consistency or limits.
- The parameters for position control can be checked and modified by accessing the structure parameter.position_control in addition.

Parameter for position control	Description
KP	Proportional gain in positive direction. Used directly to multiply the following error and create the Reference_Prop.
KF	Feed forward in positive direction. Used directly to multiply the speed reference and create the Reference_FF.
KP_BACK	Proportional gain in negative direction. Used directly to multiply the following error and create the Reference_Prop.
KF_BACK	Feed forward in negative direction. Used directly to multiply the speed reference and create the Reference_FF.

TI	Integration time. When parameter is used the position control loop has an additional integral part. In TI cycle, the Reference_ITG will reach the value of Reference_Prop, when $KI=100*KP$.
KI	Proportional gain, used for integral part of position control loop.
KF_100	Value for feed forward gain, if 100% would be used.
Max_Time	Delay time used for supervision of velocity. With Max_Time=0, no supervision is executed.
D_XS_Max	Maximum possible velocity in [u/cycle]. The maximum allowed following error is part of the parameter structure, PLCopen parameter paraMaxPositionLag.
Ref_Max	Limit for Speed_Reference.

Element actual of Axis_Ref

The element `actual` represents actual values from inside the position control loop.

Value	Description
Position	Actual position in [u] to control the axis.
Control_Position	Reference position in [u] which is actually used for control loop.
D_XS	Distance in [u] to be moved per cycle.
D_XSS	Following error in [u].
Reference_Prop	Proportional part for Speed_Reference.
Reference_FF	Feed forward part for Speed_Reference.
Reference_ITG	Integral part for Speed_Reference.

Note: It is possible to use different gains for forward/ backward movement, to see improvements for example for hydraulic axis or vertical movement

See parameter KP/KP_BACK and KF/KF_BACK.

Limitation for velocity, acceleration, and deceleration

From library version 3.1 on, these values are not limited to the 16-bit value range (32767). The limit for velocity is calculated by the values given at CMC_Axis_Control_Parameter and the acceleration is limited such that this velocity cannot be reached faster than 1 cycle.

10.1.4.4 Programming guidelines

To achieve the best results for Motion Control the actual position must be transferred in best quality (with minimal jitter) to the PLC. The position feedback is expected to be in increments as the data type is DINT.

A larger part of the below guidelines is given here for reference, but partly taken care of, if the Motion Solution Wizard (see chapter 8.1.2) is used, who automatically configures the Kernel and Parameter blocks and the associated task already in the proposed way as below (and is by default hidden – there is a box to uncheck and see the generated POU's).

The kernel Function Block (CMC_Basic_Kernel or OBIO_PTOMotionKernel or OBIO_PWMMotionKernel) must be called every cycle and its task requires a fixed cycle time.

A variable of type Axis_Ref is used to connect to the PLCopen Function Blocks and their kernelFunction Block.

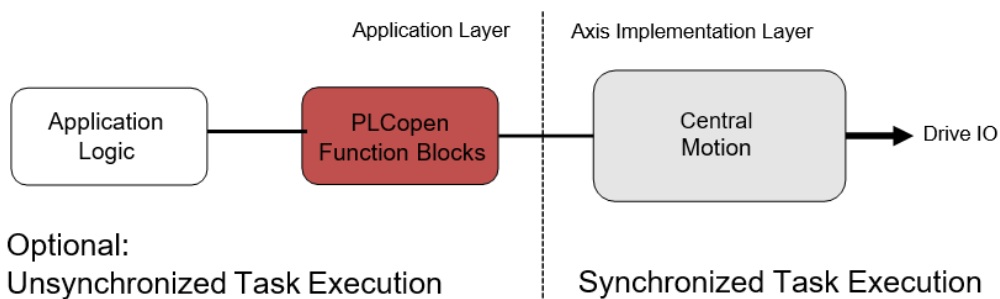
The Function Block CMC_Axis_Control_Parameter must be used for the axis configuration. Chapter “Axis parameters”.

The signal of the limits switches and the absolute switch should be connected to the elements of the data type CMC_Axis_IO. The signal of the absolute switch must be TRUE in case the axis hits the sensor. The signal of a corresponding limit switch must be true when the axis leaves the area surrounded by the limit switches. If needed the signal must be inverted before it is connected to the elements of the data type.

Task configuration

The kernel function block and the transfer of axis IO data should be processed in a cyclic task. This task should be as short and real-time as possible to achieve the best motion control performance. Always make sure Kernel function block is called at the highest priority task and other applications must be at a lower priority task.

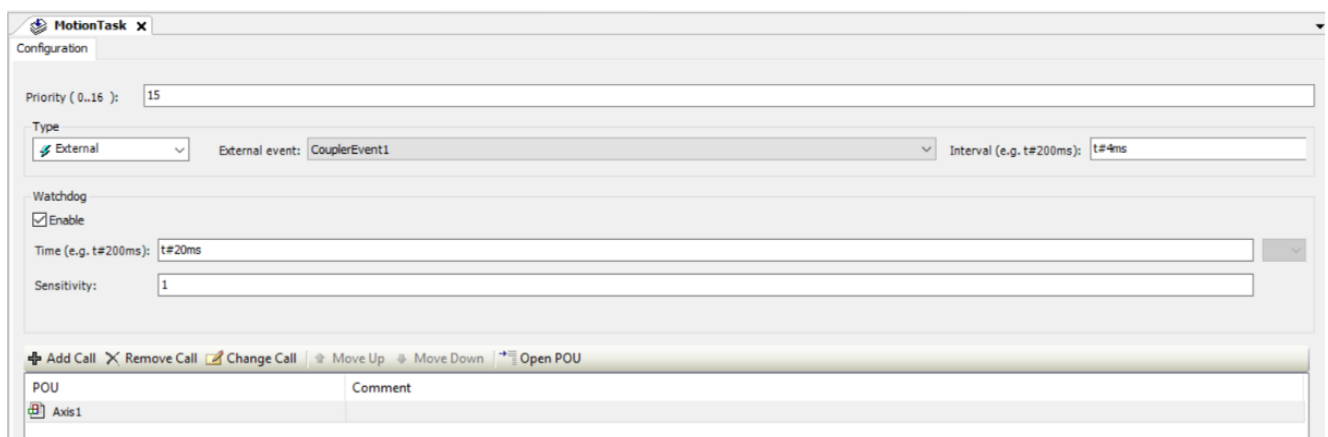
To save PLC processing time the most PLCopen function blocks as well as the application logic can also be processed in a task which runs on a lower priority than the real-time task with the axis implementation as shown in the figure below.



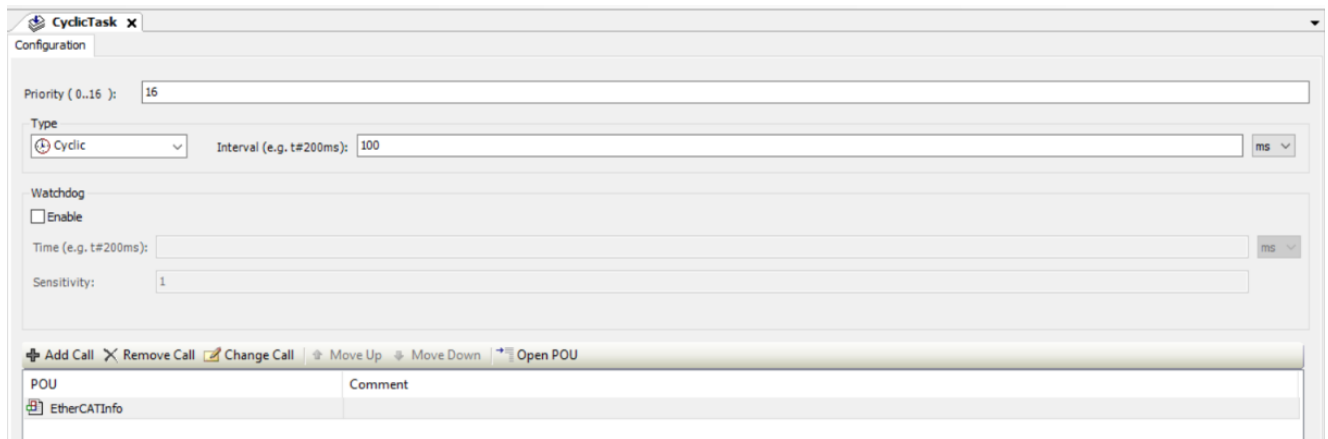
All PLCopen function blocks which must be called in the same task as the kernel function block:

- MC_CombineAxes
- MCA_MoveByExternalReference
- MCA_MoveByExtRefRelative

In case the position reference is transferred to the drive the task of the axis implementation should be synchronized to the fieldbus cycle. The following figures show an example for EtherCAT:



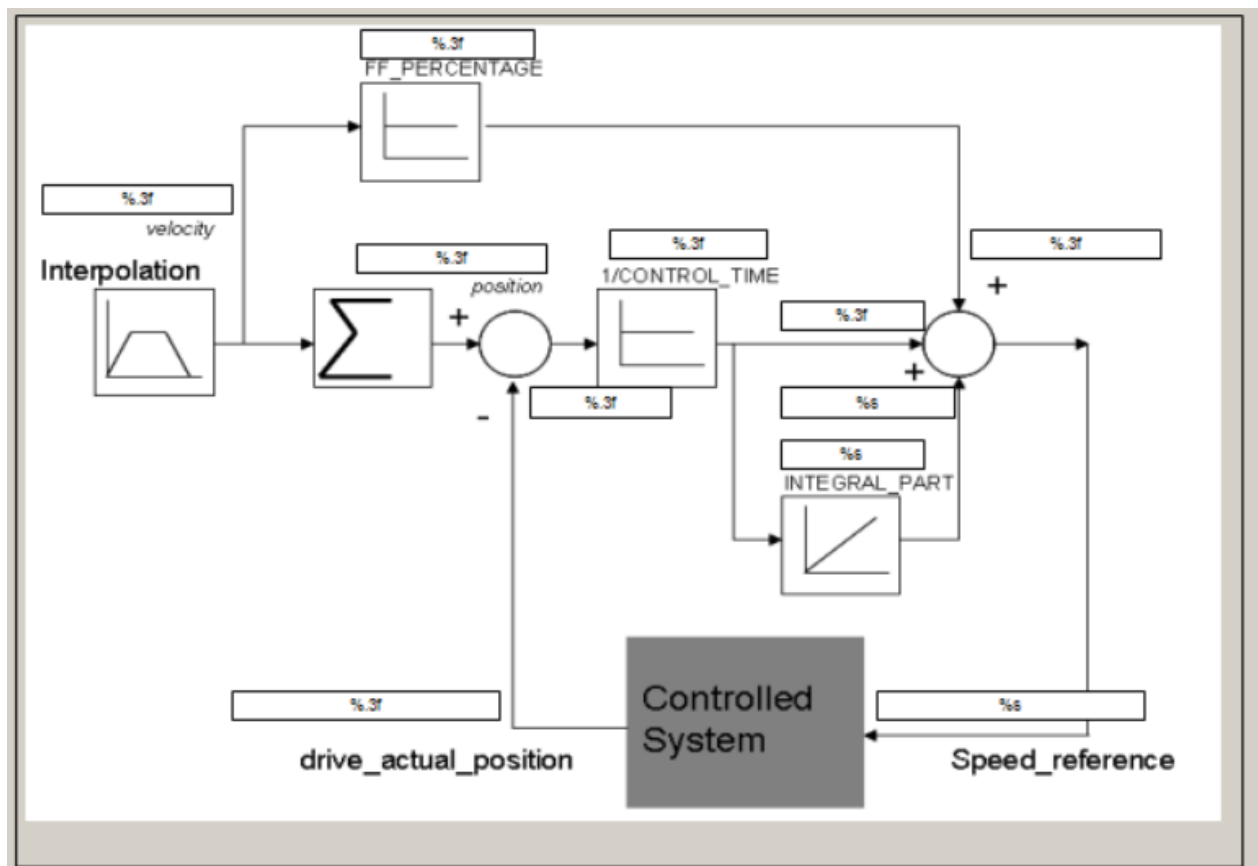
Task of axis layer



Task of application implementation

10.1.4.5 Visualization

The structure of the position control loop is also as visualization element CMC_Visu_FB_Basic_Kernel. included in ABB_MotionControl_AC500.library. As placeholder, an instance of CMC_Basic_Kernel must be used. The visualization shows all numbers as they are really used inside the block, the adjustment for different resolution or cycle times is already included.



10.1.4.6 ABB specific data structures

Not all data structures are defined by PLCopen. Some specific structures are described in the following chapter. In addition to the data in these arrays, the movement is modi-

fied by offset and scaling values at the respective Function Block. These offset and scaling values (except the timescale) are transferred continuously. This will allow us to follow a "Moving Target" by adjusting these values.

10.1.4.6.1 PositionPositionProfile

The data type MC_PProfile is used for CamTable. An array has to be defined and provided at MC_CamTableSelect. Several CamTables could be defined, and the axis could change between them on the fly. There is no routine of smooth movement from one table to the next, so the user must take care just to switch on appropriate positions. Details are described in the documentation included with the library.

Declaration example CAM_table

ARRAY[1..3] OF MC_PProfile:=

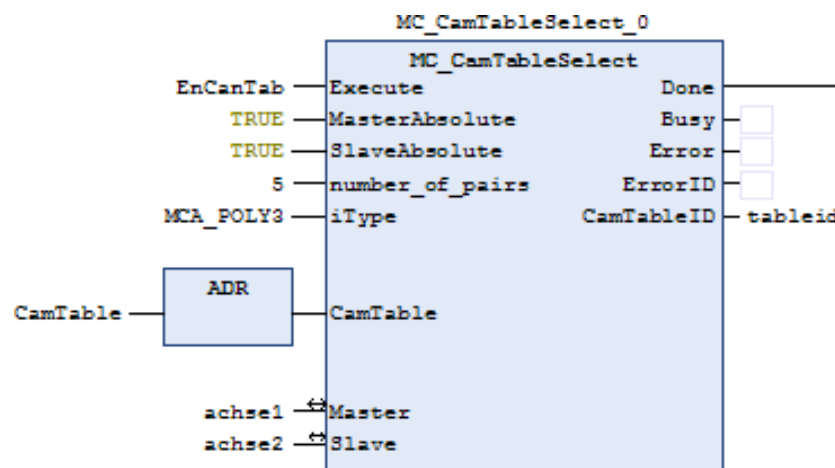
```
(Master_position:= 0 ,interpolation_point := 0 ,Velocity_ratio:= 0 ,Acceleration_ratio:= 0 ),
(Master_position:= 50 ,interpolation_point := 25 ,Velocity_ratio:= 0 ,Acceleration_ratio:= 0 ),
(Master_position:= 100 ,interpolation_point := 0 ,Velocity_ratio:= 0 ,Acceleration_ratio:= 0 );
```

10.1.4.6.2 PositionTimeProfile

This structure is used for time-based profiles, e.g., MC_PositionProfile:

10.1.4.6.3 Interpolation types for profiles

The curves defined by an array of MC_PProfile hold master position points and according to slave positions. When the master position is between 2 points, the according position for the slave is interpolated. Different types of interpolation are possible. The type is defined in MC_ABB_iTypes_Enum . The master could be a real axis or some virtual axis which could be created by just writing values for position and velocity to the Axis_Master variable as shown in the example. The same interpolation types could be used on MC_TProfile.



Overview of different interpolations

Interpolation Types	Results in	Requires
MCA_Linear	Linear interpolation with constant velocity between interpolation points.	profile.MC_PProfile_Array[x].master_position,profile.MC_PProfile_Array[x].interpolation_point

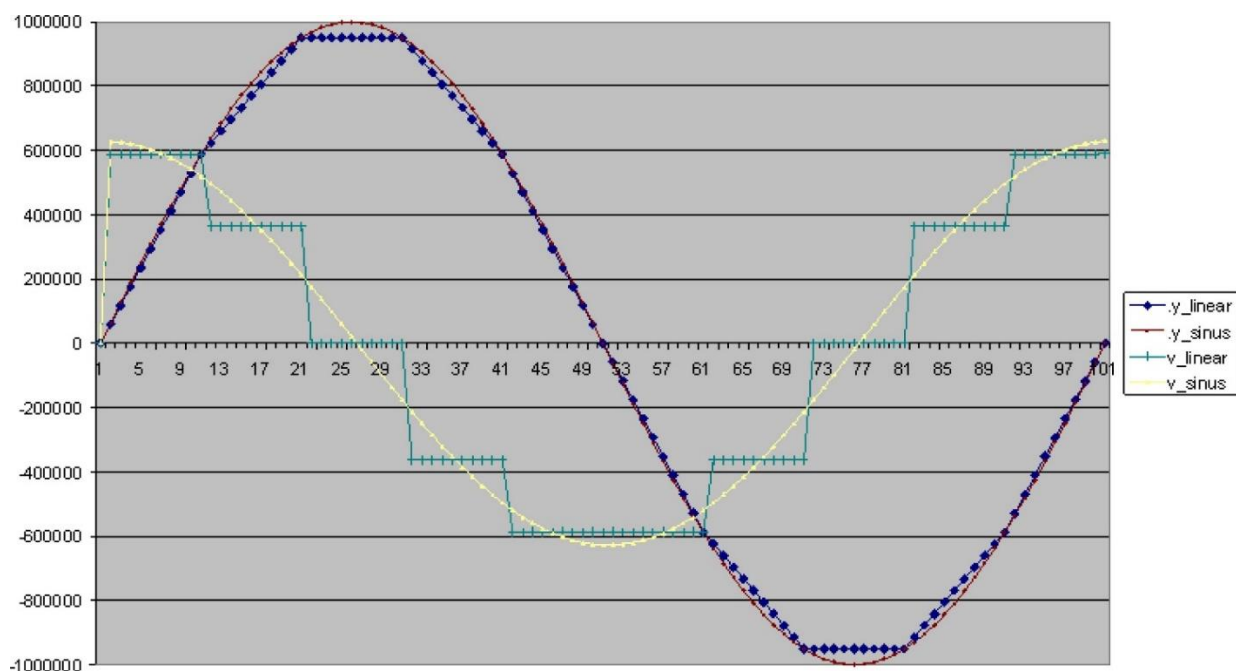
MCA_Spline_Natural	Cubic spline interpolation without jerk.	profile.MC_PProfile_Array[x].master_position, profile.MC_PProfile_Array[x].interpolation_point
MCA_Spline_Complete	Cubic spline interpolation without jerk, start and end of profile with velocity=0.	profile.MC_PProfile_Array[x].master_position, profile.MC_PProfile_Array[x].interpolation_point
MCA_Poly3	Polynomial interpolation with linear velocity between interpolation points.	profile.MC_PProfile_Array[x].master_position, profile.MC_PProfile_Array[x].interpolation_point, profile.MC_PProfile_Array[x].velocity_ratio
MCA_Poly5	Polynomial interpolation with linear acceleration between interpolation points.	profile.MC_PProfile_Array[x].master_position, profile.MC_PProfile_Array[x].interpolation_point, profile.MC_PProfile_Array[x].velocity_ratio, profile.MC_PProfile_Array[x].acceleration_ratio

The interpolations allow to run on smooth curves without the need to define a large number of points. The following chapter shows the results with different interpolation modes for a sinus-curve with 10 interpolation points. The following table gives the mean deviation.

Interpolation Type	Mean deviation [ppm]
MCA_Linear	19686 =1.9%
MCA_Spline_NATURAL	151=0.0151%
MCA_Spline_Complete	25510=2.5%
MCA_Poly3	131=0.0131%
MCA_Poly5	0.37

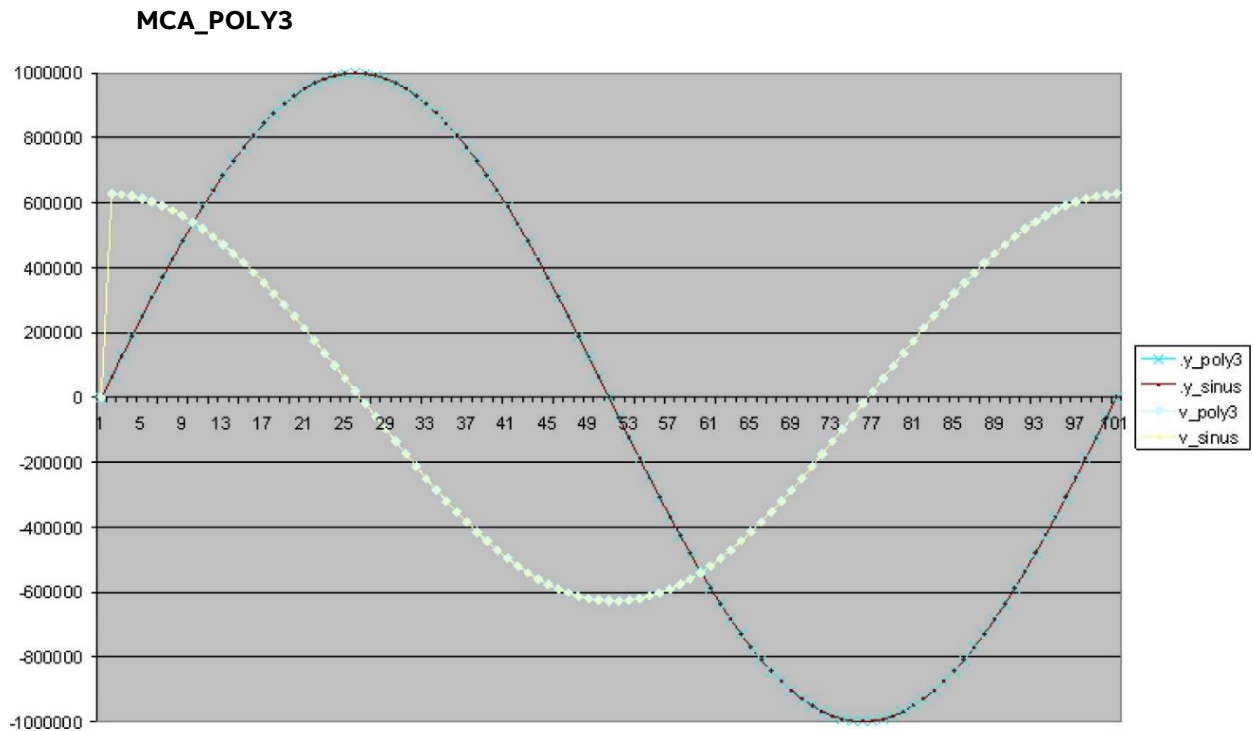
The original curve is represented by y_sinus for position and v_sinus for velocity. The diagrams show the result which is achieved by different interpolation types.

MCA_LINEAR



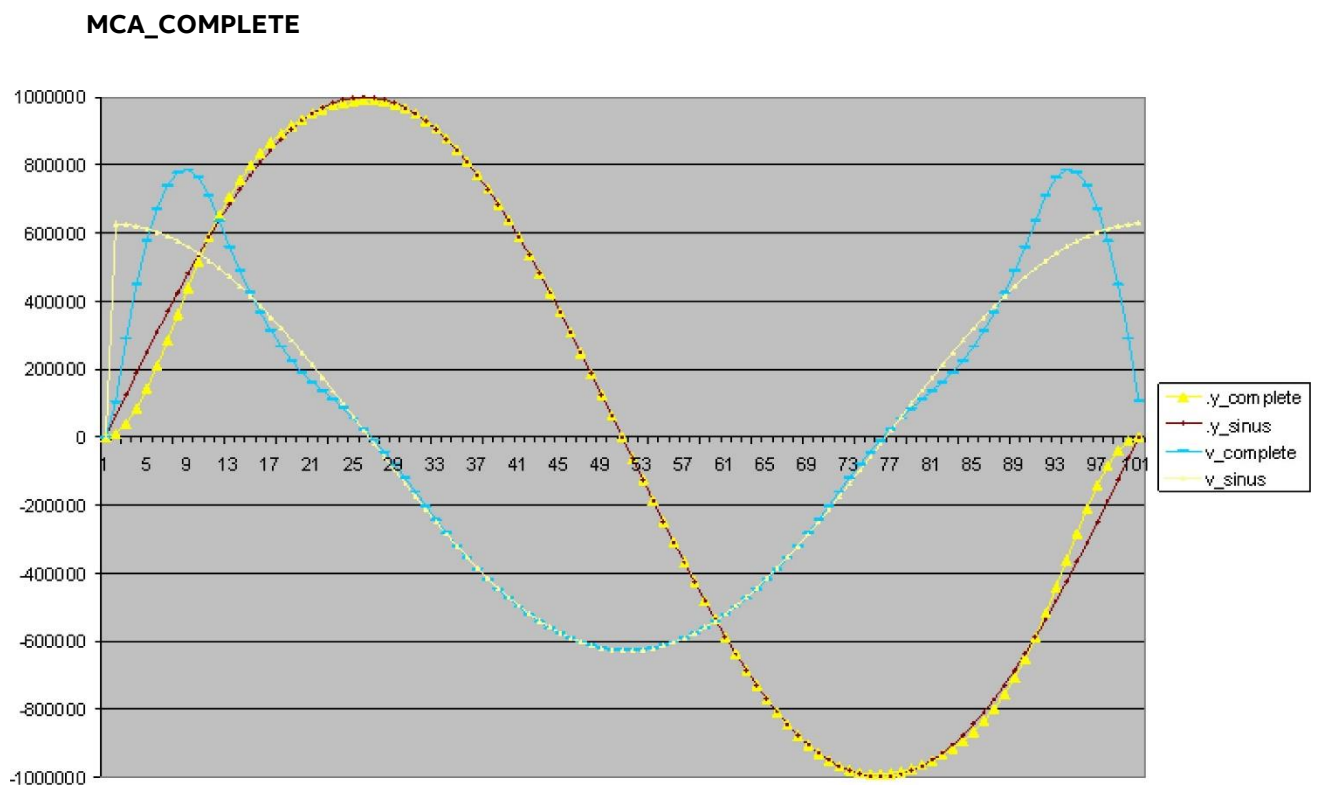
Results from linear interpolation

The velocity is constant between the interpolation points.



Results from polynomial interpolation

The result looks almost identical to the original curve. The mean deviation shows that MCA_POLY3, MCA_POLY5 and MCA_SPLINE_NATURAL produce results which follow the original curve really good and are almost identical. The spline interpolation produces a jerk-free curve without the need of providing velocity values and acceleration values in advance.



Results from complete spline interpolation

In the beginning and the end, the curve does not follow the original curve. The reason is that it starts with velocity=0 and produces a jerk free result.

So the favoured result has to be considered in advance to choose the right interpolation method. With these different methods it is not necessary to provide a large number of interpolation points to get good results and smooth acceleration and deceleration ramps.

10.1.5 Load Control/Torque Control: Fluid Power Extension according PLCopen

The ABB_MotionControlLoad_AC500 library is an extension to ABB_MotionControl_AC500 library based on PLCopen part 6 called “fluid power” and can be used to implement load control as a simple form of torque profiling.

It can be used together with all other motion control package libraries (but due to its nature of course NOT with stepper motors/the eCo kernel library). The same structure and general rules are applied and all the above chapters in this document is relevant for ABB_MotionControlLoad_AC500 library as well. A difference is that the position control loop has to be closed inside the PLC as it is to be synchronized with the load control loop which is also realized.

Overview of the defined extended Function Blocks:

Administrative	Motion
Single Axis	
MC_LimitLoad	MC_LoadControl
MC_LimitMotion	MC_LoadSuperImposed
	MC_LoadProfile
	MC_TorqueControl

Note – As per PLC open MC_TorqueControl is a part1 function block, however due to its implementation as a wrapper for the load control and limit load blocks this is added to ABB_MotionControlLoad_AC500 library.

The following state diagram is based on the version as defined in ‘Part 1 – Function Blocks for Motion Control’, Version 2.0

This specification adds three Load Function Blocks to the State Diagram:

- MC_LoadControl
- MC_LoadSuperImposed
- MC_LoadProfile

MC_TorqueControl function block also follows the same state diagram.

Function Blocks not listed in the state diagram do not affect the State Diagram, meaning that whenever they are called the state does not change.

The State diagram shows Synchronized Motion because the position-axis follows the load, and the state is related to the position axis

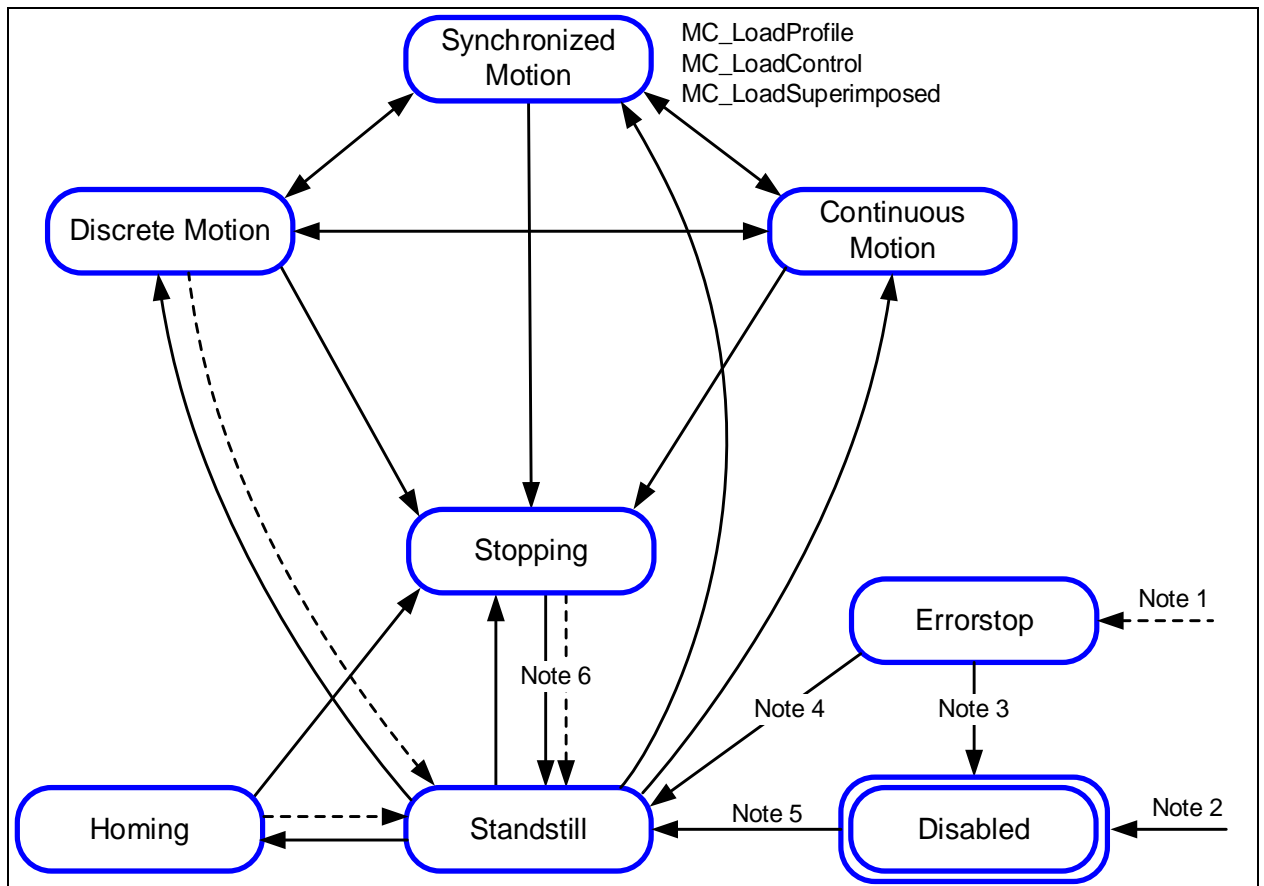
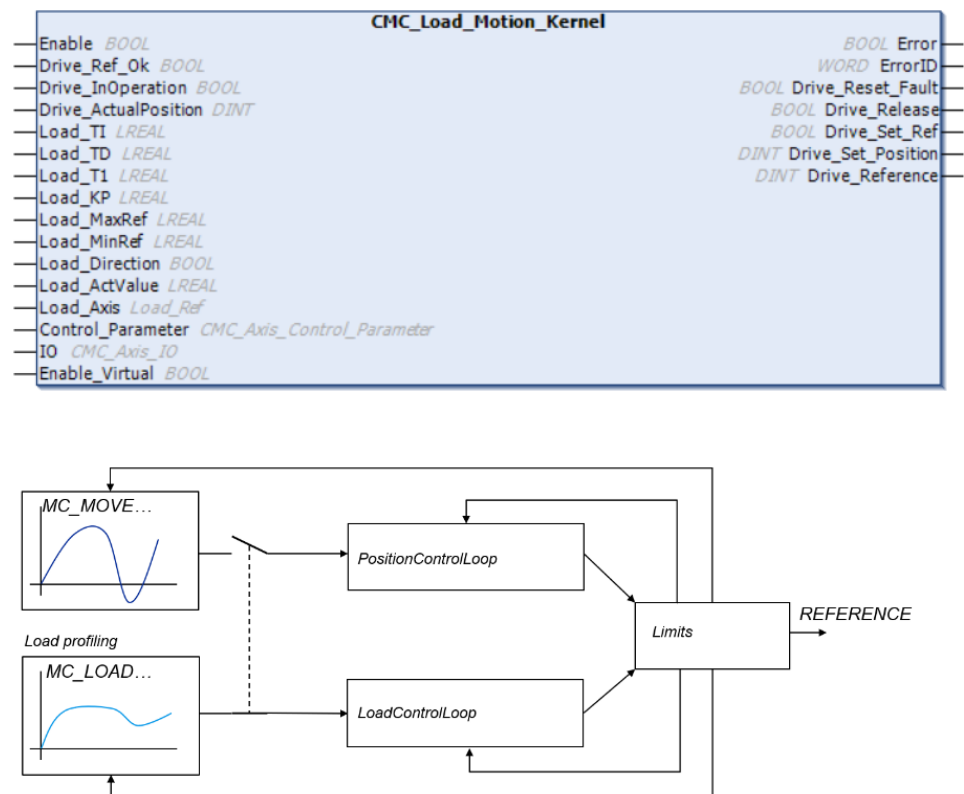


Figure: The State Diagram

Kernel function block - Fluid Power

The basic block is the CMC_Load_Motion_Kernel. It has to be called every cycle and at least once before any MC... block is activated. It is used to combine the position and velocity functionality from CMC_Basic_Kernel with the load control functionality which is utilized by the MC_Load.. blocks.



The reference which is used by the CMC_Load_Motion_Kernel is equivalent with the Speed_Reference at CMC_Basic_Kernel, as long as no LOAD-functionality is activated.

The documentation from CMC_Basic_Kernel applies to the identical inputs and outputs. Some inputs and outputs are added to serve the load control functionality.



Note: The Load_Ref is used instead of Axis_Ref for the MC_Loadxxx blocks.

When the CMC_Load_Motion_Kernel is used, Load_Ref replaces Axis_Ref and user can use all PLCopen-Blocks.

The actuator (drive) has to be accessed outside the CMC_Load_Motion_Kernel block. actual values and reference values might be transferred by a synchronised bus or by I/Os.

All inputs and outputs of the function block which are named “DRIVE_xxxx” should be used to connect to the actuator (drive). It does not matter whether this connection is done by fieldbus or by conventional I/Os.

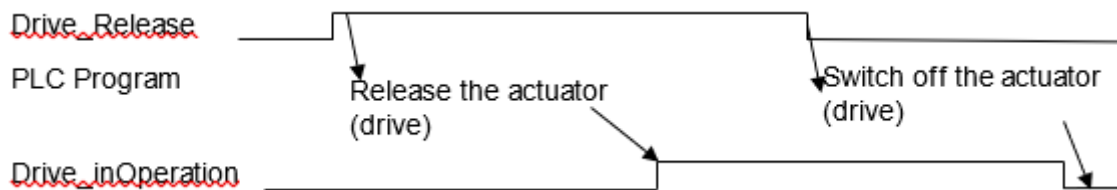
The Axis-structure is used to connect to the PLCopen Blocks

The Load_Axis structure is used to connect the fluid-power PLCopen blocks

The control_parameter-structure is used for configuration of control loop.

The IO-structure gives a connection to limit- or reference switches.

When the function block will take control (close loop) the output “Drive_Release” is set. The PLC-Program should then start the actuator (actuator (drive)) and set “Drive_InOperation = TRUE” when successful. In case of actuator (actuator (drive)) problem, “Drive_InOperation” should be reset. The function block will then open the position control loop and Speed_Reference will be 0.



The homing is done with PLCopen-Blocks. As the interface to the actual position is outside the CompactMotion, the bit “Drive_Set_Ref” is set when the state is reached to evaluate the zero-track. When the zero-track was found, Drive_ActualPosition has to be set to “Drive_Set_Position”, this has to be indicated by “Drive_Ref_Ok”.

The output “Drive Reference” should be send to the actuator (drive). This value is scaled with Max_Rpm and Max_Reference which means: when “Drive_Reference” equals Max_Reference, the motor is expected to run with Max_Rpm.

Load Control

The function block holds a position control loop and a load control loop. The load control loop is a PIDT1-Block. Both control loops are alternately activated, depending if a MC_Load..block or a MC_Move... block is active. There is a bumpless transition realized between the different control loops.

The PIDT1 controller has a proportional, integral and derivative part. The integral and derivative part can be switched of by using a time value = 0.

Transfer function

$$F(s) = K_P \cdot \left(1 + \frac{1}{s \cdot T_N} + \frac{s \cdot T_V}{1 + (s \cdot T_I)} \right)$$

Control algorithm: Simple rectangle rule:

$$Y = \frac{K_P \cdot X_D}{100} + \frac{K_P}{100} \cdot \frac{X_D}{T_N \cdot T_Z} + Y_I(z-1) + \frac{T_I \cdot T_Z}{1 + (T_I \cdot T_Z)} \cdot (Y_{DTI}(z-1) + \frac{1}{T_I \cdot T_Z} \cdot \frac{T_V}{T_Z} \cdot \frac{K_P}{100} \cdot (X_D - X_D(z-1)))$$

Where:

$Y_I(z-1)$: The integral portion from the previous program cycle

$Y_{DTI}(z-1)$: The differential portion from the previous program cycle

$X_D(z-1)$: Control system difference from the previous program cycle

All 3 parts of the control loop are added up. The integral or derivative part could be disabled by setting the respective time constant to 0, so the following structures are possible:

- P
- PDT1
- PI
- PIDT1

The Load_MaxRef and Load_MinRef values will limit the controllers output Y and also apply to the controller's internal integral part. I.e the integral part can only hold values between the high and low limits. If the manipulated variable Y reaches one of the two limits, the controller's integral part is no longer changed. This prevents the integral part from holding meaningless values and, in certain circumstances, not returning to the operating range for a long time. This behavior of a controller is also referred to as a »special anti-reset windup measure.

Example - Fluid Power Extensions

MC_LimitLoad Example

In the diagram below, an example is explained. SFC is used here to distinguish between a movement where the MC_LimitLoad functionality has become 'Active' or not. In Step 2 there is a movement like 'MoveAbsolute', which is limited by the MC_LimitLoad functionality. If the absolute position is reached without MC_LimitLoad becoming active, the transition via done to step 3 is applicable. However, if the MC_LimitLoad becomes 'Active', the transition to the 'Halt' step is applicable, issuing a MC_Halt.

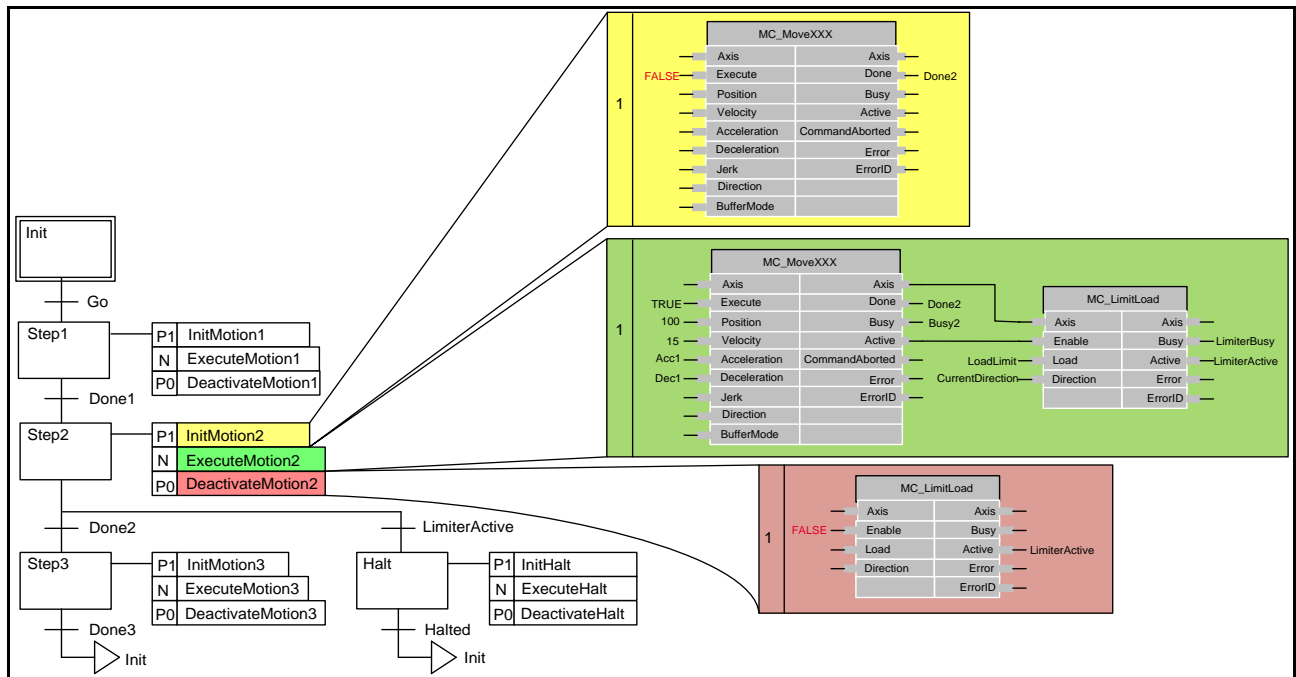


Figure: MC_LimitLoad used in SFC

MC_LimitMotion Example e.g. force fitting.

The FB is intended to be used in conjunction with a MC_LoadControl or MC_TorqueControl having primary control on the axis. The MC_LimitMotion should be enabled by the 'Active' output of the MC_LoadControl / MC_TorqueControl. If motion values on the axis exceed the given limit, appropriate measures are taken to keep to these limits, implying that the load/torque will not follow the programmed trajectory but depend on the external load conditions. However, the 'Active' output of the MC_LoadControl/MC_TorqueControl will stay TRUE in this case, following the modified PLCopen definition "The 'Active' output indicates, that the FB has control on the set-value generation of the axis". This is despite the fact, that physically only the load-conditions or the movement of an axis can be controlled. With actual motion states below programmed limits, the programmed load/torque trajectory will proceed. Enabling the limiter block with activation of the MC_LoadControl/MC_TorqueControl ensures that limits are only supervised when the MC_LoadControl/MC_TorqueControl takes control on the axis for the first time. Disabling the limiter block with de-activation of the MC_LoadControl/MC_TorqueControl ensures that limits are no more supervised when the MC_LoadControl/MC_TorqueControl loses control on the axis by 'CommandAborted' or 'Error'

MC_LoadSuperImposed Example

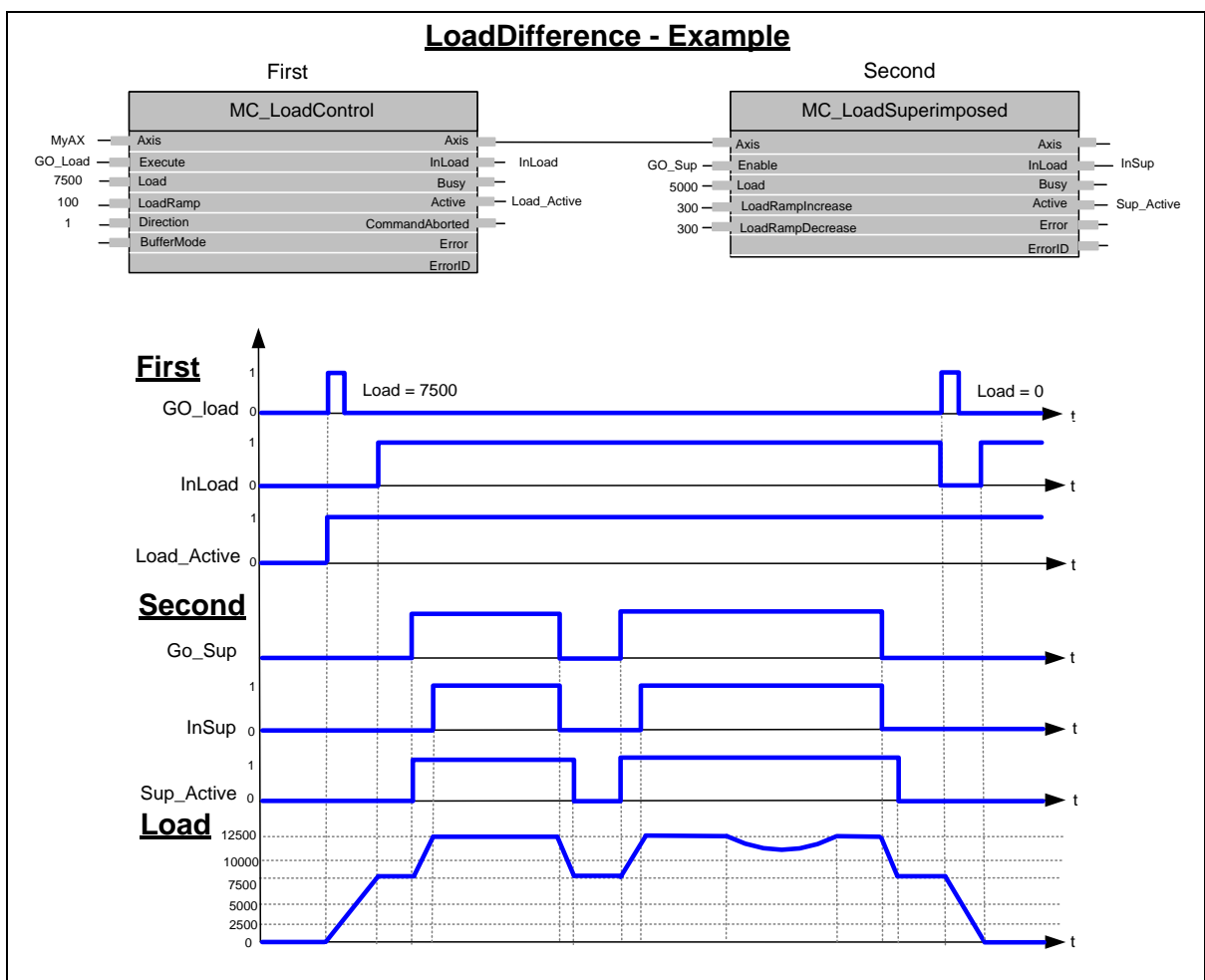
Possible Application: Actuator: hydraulic cylinder with fluid pressure sensor actuates the press of plastic injection molding machine in a continuous load operation.

Request: prior to MC_LoadSuperImposed call, a MC_LoadControl block is 'Active' with a command of 7,500 kPa to press melted plastic into the mold. Once the MC_LoadControl 'InLoad' condition is achieved a superimposed pressure of 5,000 kPa is added several times to cause a hammering effect to relieve stresses in the plastic.

Result: the MC_LoadControl pressure command of 7,500 kPa is superimposed with a discrete pressure command of 5,000 kPa. Once the 'LoadSuperImposed' command is active the system pressure rises to 12,500 kPa.

When the superimposed pressure command has been achieved the MC_LoadSuperImposed block is done and the original command given by the MC_LoadControl resumes the original pressure command.

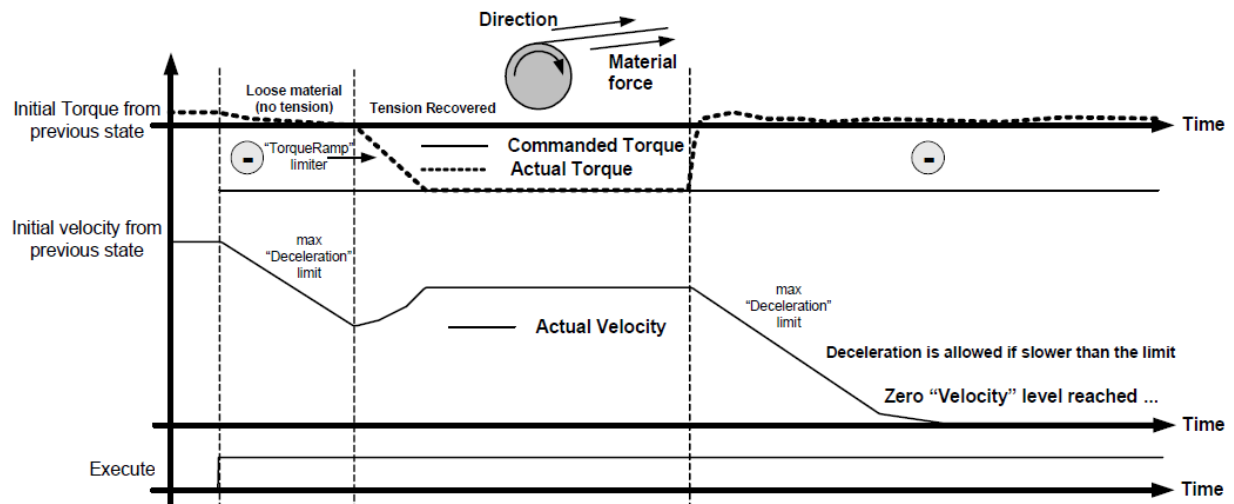
The MC_LoadSuperImposed block is executed several times without affecting the original pressure command given by the MC_LoadControl block.



MC_TorqueControl Example

The example (below) opposite signs for 'Direction' & 'Torque' are used (e.g. Retention or brake control). (In the FB: +Direction –Torque). It is like an unwinding application with torque on the material, and a break in the material. When the material breaks, as shown in the middle of the picture, this causes a drop in the real Torque value (in absolute terms): the velocity will decrease, limited by the fastest "deceleration" limit specified by the 'Deceleration' VAR_INPUT down to zero velocity (with no tension there is a risk of

having shock breakings, so we must limit to the fastest). In this case the torque setpoint might not be achieved.



NOTE: In an unwinding application (derived from this brake control) material tension is the target, not motor torque. The instantaneous diameter of the roll should be taken into account to transform the "User tension setpoint". Also, additional inertia compensation by modification of the torque setpoint for acceleration / deceleration is common from instantaneous weight data (weight is commonly estimated from diameter). Additionally, in unwinding applications, in the case of loose material (same condition as material break), a negative slow velocity reference is usually applied to "rewind" the loose material. In this case, this must be provided by external programming.

10.2 PLCopen based Motion Control Libraries (Function Block descriptions)

10.2.1 MotionControl (Library)

The PS5611 Motion Control library is to create the motion control applications according to PLCopen Motion definition. This library contains PLC open standard blocks (MC), ABB Specific (MCA) and Central Motion control (CMC) function blocks. Using these function blocks one can realize the Motion control functionalities such as: Simple axis movements, Position Profiles, Acceleration profiles, velocity profiles, Camming, GearIn, Homing and so on.

The library also contains visualization for each function block and visualization for the statemachine.

This Library needs PLC based runtime license for using the features.

Copyright: We reserve all rights in these programs and the information therein. Reproduction, use or disclosure to third parties without express authority is strictly forbidden.
(c) 2006-2021 ABB, all rights reserved

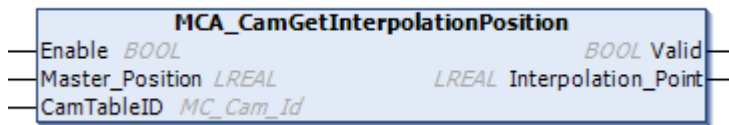
10.2.1.1 PLCopen

10.2.1.1.1 ABB Specific

ABB Specific PLC open motion control blocks. All function or function block names will start with Prefix MCA.

10.2.1.1.1.1 MCA_CamGetInterpolationPosition (FB)

This function block gives an interpolation result, according to the referenced camtable, for the given master position.



Mode: The positions will be used as absolute positions, offset and scaling will NOT be considered.

The block will not check if the cam table is still active.



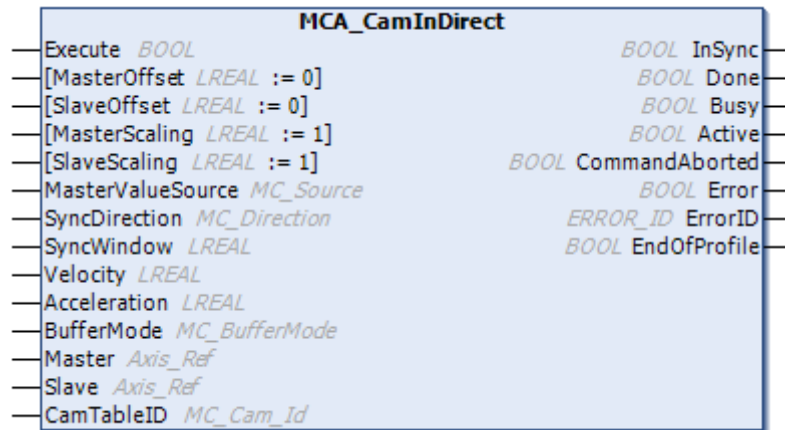
User can utilize CAM editor in Automation builder to generate Cam table (MC_PProfile) automatically. For more details refer to Automation Builder help.

InOut:

Scope	Name	Type	Comment
Input	Enable	BOOL	Activate the function block
	Master_Position	LREAL	Master position for cam table
Out-put	Valid	BOOL	FALSE if either of the table is not valid, or master position outside the range
	Interpolation_Point	LREAL	Gives the interpolated position
Inout	CamTableID	MC_Cam_Id	Prepared by MC_CamTableSelect and used with MC_CamIn or MCA_CamInDirect

10.2.1.1.1.2 MCA_CamInDirect (FB)

This function block implements Camming-Functionality. A slave axis is coupled to a master axis by a position/position relation.



- It is not required that the master is stationary.
- If the actual master and slave positions do not correspond to the offset values when MC_CamIn is executed, either an error occurs or the system deals with the difference automatically.
- The Cam is placed either absolute or relative to the current master and slave positions:
- Absolute: The profile between master and slave is seen as an absolute relationship.
- Relative: The relationship between master and slave is in a relative mode.
- If a cam-table is to be used “relative”, the first position has to be zero.
- This function block is not merged with the MC_CamTableSelect function block because this separation enables changes on the fly.
- A mechanical analogy to a slave offset is a cam welded with additional constant layer thickness. Because of this the slave positions have a constant offset and the offset could be interpreted as axis offset of the master shaft, if linear guided slave tappets are assumed.








The Slave axis is not ramped out, which means the curve should end with velocity = 0. The CAM could be interrupted with any other function block, according to the statemachine. It is not required to use MC_CamOut.

The function block behaves as follows:

- If the master is inside the position range which is described in the cam-table data, synchronization starts right away, no matter if the master moves or is in stand-still.
- If the master is outside the position range which is described in the cam-table data, the slave position is not modified.
- The synchronization is limited by the given Velocity and Acceleration, achieved as fast as possible. The function block will show InSync when synchronization process is completed and the slave axes reference position matches the cam-table data for the current master position.

- In a modulo-axis, it is possible to reach the synchronization point in different directions. The input parameter SyncDirection with its possible values: POSITIVE, NEGATIVE or SHORTEST can be used to set this direction. Inside the SyncWindow, automatically the direction SHORTEST will be used.
- It is important to set a SyncWindow > 0 for a modulo axis, otherwise, slightest deviations could result in moving a complete modulo distance.
- Inside SyncWindow, the slave axis will move SHORTEST to reach the SlaveSyncPosition
- Outside SyncWindow, it will move the given SyncDirection, which can be POSITIVE or NEGATIVE
- If a direction POSITIVE or NEGATIVE is used in a linear axis, the slave will wait until the master reaches position which allows the slave to move the required direction.

	The MCA_CamInDirect has parameters to scale the cam-table values (MasterScaling, SlaveScaling). It has to be considered that MasterOffset and SlaveOffset are scaled exactly like the corresponding cam-table values. The MasterSyncPosition and MasterStartDistance are not scaled at all, these positions are related to the actual master position whereas the MasterOffset and SlaveOffset are related to the camtable.
	New set of values at inputs MasterOffset, SlaveOffset, MasterScaling, SlaveScaling will be accepted only after the function block is aborted and fresh rising edge is provided at Execute input.
	The default behavior of this function block can be modified by the inputs in function MCA_Cam_Extra
	A negative MasterScaling requires backward master movement, when combined with MasterOffset. The MasterScaling also applies to the MasterOffset. Behaviour results from the requirement to have ascending master values in CamTable.
	User can utilize CAM editor in Automation builder to generate Cam table (MC_PProfile) automatically. For more details refer to Automation builder help.

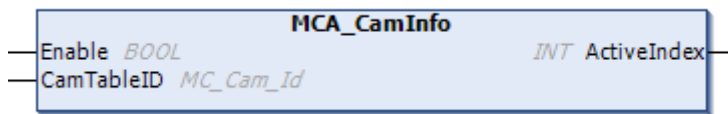
InOut:

Scope	Name	Type	Initial	Comment
Input	Execute	BOOL		Starts the Function Block at rising edge
	MasterOffset	LREAL	0	Offset of master table. Actual position - MasterOffset will be used to sample the CamTable
	SlaveOffset	LREAL	0	Offset of slave table. Sharpened cam (i.e higher elevation and deeper depression). Use the result from CamTable + SlaveOffset as reference position
	MasterScaling	LREAL	1	Scaling factor for master positions in CamTable. From the slave point of view the master overall profile is multiplied by this factor

Scope	Name	Type	Initial	Comment
	SlaveScaling	LREAL	1	Scaling factor for slave positions from CamTable. The overall slave profile is multiplied by this factor
	MasterValueSource	MC_Source		Defines the source for synchronization: mcSetValue - Synchronization on master set value. mcActualValue - Synchronization on master actual value
	SyncDirection	MC_Direction		Moving direction for the slave to start the movement. Applicable: POSITIVE or NEGATIVE, use SHORTEST for any other value
	SyncWindow	LREAL		[u], Used to determine the moving direction, combined with SyncDirection When the slave is outside the SyncWindow, it will move the direction which is given in SyncDirection When the slave is inside the SyncWindow, it will move SHORTEST to meet the SlaveSyncPosition
	Velocity	LREAL		[u/s] Used for Synchronization. Range: ≥ 0 , max application velocity (Parameter9) used as default The slave has to be able to move faster than the master axis, otherwise it is possible the SlaveSyncPosition is never reached when the master starts to move
	Acceleration	LREAL		[u/s ²] Used for Synchronization. Range: ≥ 0 max application acceleration (Parameter13) used as default
	BufferMode	MC_Buffer-Mode		Not supported, default mcABORTING used
Output	InSync	BOOL		Slave is synchronized to CamTable
	Done	BOOL		Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BOOL		The function block is not finished
	Active	BOOL		Indicates that the function block has control on the axis
	Command-Aborted	BOOL		Command is aborted by another command from other PLCopen function block
	Error	BOOL		Signals that error has occurred within function block
	ErrorID	ERROR_ID		Error identification. For error details refer to Enumeration ERROR_ID
Inout	EndOfProfile	BOOL		Pulsed output signaling the cyclic end of the CAM profile. It is displayed every time when the end of CAM profile is reached. In reverse direction, the 'EndOfProfile' is displayed also at the end of the cam profile (in this case the first point of the cam profile)
	Master	Axis_Ref		Reference to master axis
	Slave	Axis_Ref		Reference to slave axis
	CamTableID	MC_Cam_Id		Prepared by MC_CamTableSelect. Identifier of CAM Table to be used in the MC_CamIn Function Block

10.2.1.1.1.3 MCA_CamInfo (FB)

This function block gives an information which index is actually processed by the respective cam table.



Precondition: Correct information is shown if the referenced cam table is still active.

The block will not check if the cam table is still active.



User can utilize CAM editor in Automation builder to generate Cam table (MC_PProfile) automatically. For more details refer to Automation builder help.

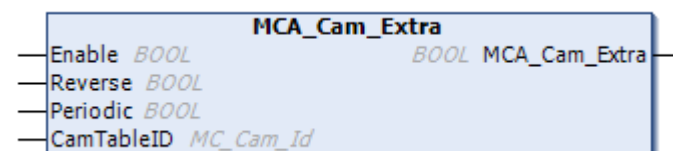
InOut:

Scope	Name	Type	Comment
Input	Enable	BOOL	Activate the function block
Inout	CamTableID	MC_Cam_Id	Prepared by MC_CamTableSelect and used with MC_CamIn or MCA_CamInDirect
Output	ActiveIndex	INT	Active Index, value always starts from zero, even if the CAM table array is started with a different value

10.2.1.1.1.4 MCA_Cam_Extra (FUN)

This function is just usable together with MC_CamTableSelect and should be called right after MC_CamTableSelect to modify 2 mode-bits which define the behavior for the MC_CamIn more precise.

Without this function, the default values will be used instead to configure some additional flags for cam table behavior.



This function modifies the CAM Table behavior

- With Enable = TRUE, two bits will be written all the time and will be effective. So a Cam table could be used in Periodic = TRUE mode and will come to a stop when the master leaves its position range when Periodic = FALSE is used.
- With MODULO-AXIS: Usage with Periodic = FALSE and position range for master equals MODULORANGE: When the master reaches 360°, the movement will be ready, even when it was started just at 359°.



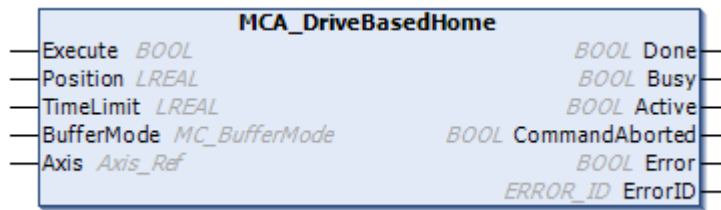
User can utilize CAM editor in Automation builder to generate Cam table (MC_PProfile) automatically. For more details refer to Automation builder help.

InOut:

Scope	Name	Type	Comment
Return	MCA_Cam_Extra	BOOL	
Input	Enable	BOOL	Activate the function
	Reverse	BOOL	Cam should be run in the reverse direction, master axis will move from larger to smaller positions when the cam table is entered
	Periodic	BOOL	Cam table will run continuously and start automatically again, if "EndOfProfile" is reached
	CamTableID	MC_Cam_Id	Prepared by MC_CamTableSelect. Identifier of CAM Table to be used in the MC_CamIn function block

10.2.1.1.1.5 MCA_DriveBasedHome (FB)

This function block can be used to execute a homing procedure directly in the drive.



Precondition: It requires the drive supports 402-profile specific homing sequences.

The function block can be used in combination with:

- ECAT_402ParameterHoming_APP to send parameters
- ECAT_CiA402_Control_App to control the drive statemachine and to set it to the appropriate operating mode



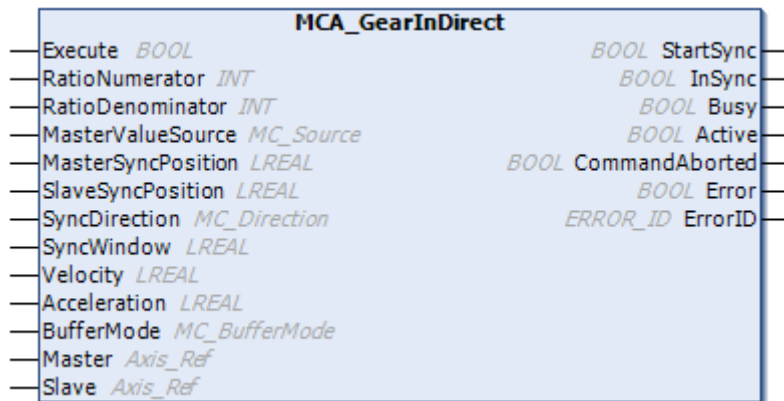
Homing is based on the settings in drive and drive has to execute the homing algorithm.

InOut:

Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
	Position	LREAL	[u] Reference position, used as position value at reference position
	TimeLimit	LREAL	[s] A time in seconds, which will be used as an upper limit for the time available to do the homing. If the time is exceeded, the function block will show an Error. With TimeLimit = 0, the limit is ignored
	BufferMode	MC_BufferMode	Not supported, default mcABORTING used
Output	Done	BOOL	Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BOOL	The function block is not finished
	Active	BOOL	Indicates that the function block has control on the axis
	CommandAborted	BOOL	Command is aborted by another command from other PLCopen function block
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
Inout	Axis	Axis_Ref	Reference to axis

10.2.1.1.1.6 MCA_GearInDirect (FB)

This function block commands a gear ratio between the position of the slave and master axes from the synchronization point onwards.



The function block behaves as follows:

- Synchronization starts right away, no matter if the master moves or is in stand-still.
- The synchronization is limited by the given velocity and acceleration, achieved as fast as possible, so it can happen that:
- The 2 axes are synchronized earlier than the two given positions
- The 2 axes are synchronized later than the two given positions
- Following formula is used:
- $\text{SlavePosition} = (\text{masterPosition} - \text{MasterSyncPosition}) * \text{RatioNumerator} / \text{RatioDenominator} + \text{SlaveSyncPosition}$
- In a modulo-axis, it is possible to reach the synchronization point in different directions.

The input parameter SyncDirection with its possible values: POSITIVE, NEGATIVE or SHORTEST can be used to set this direction. Inside the SyncWindow, automatically the direction SHORTEST will be used. It is important to set a SyncWindow > 0 for a modulo axis. Otherwise, slightest deviations could result in moving a complete modulo distance.

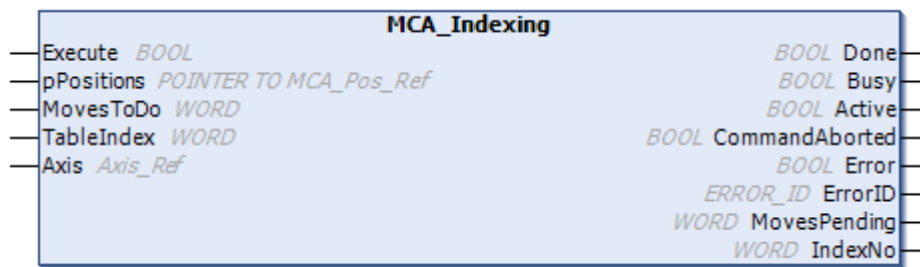
- Inside SyncWindow, the slave axis will move SHORTEST to reach the SlaveSyncPosition
- Outside SyncWindow, it will move the given SyncDirection, which can be POSITIVE or NEGATIVE
- If a direction POSITIVE or NEGATIVE is used in a linear axis, the slave will wait until the master reaches a position which allows the slave to move the required direction.

InOut:

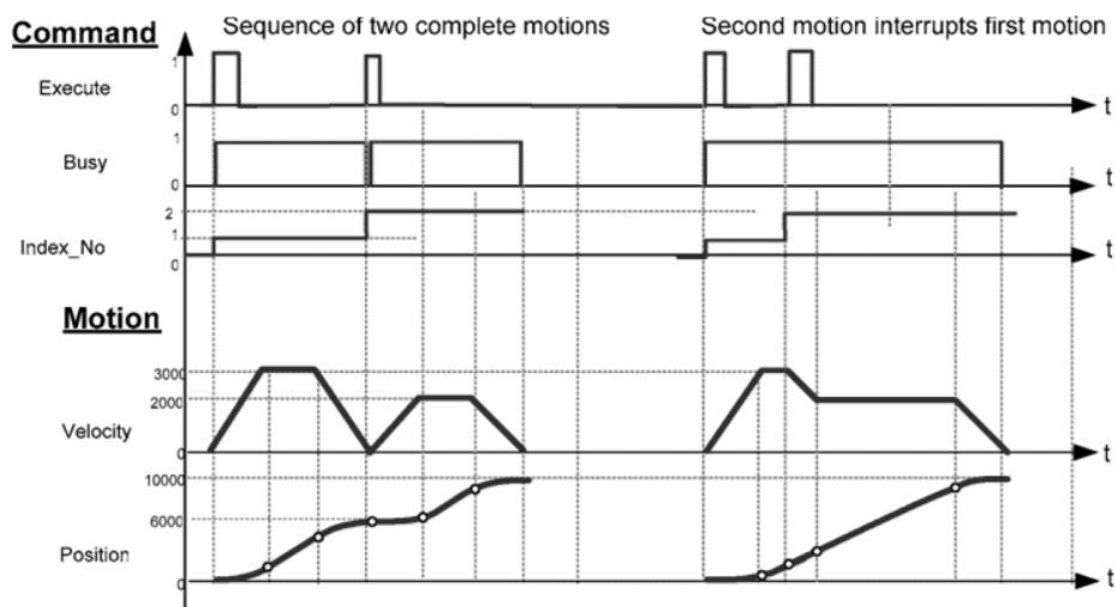
Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
	RatioNumerator	INT	Gear ratio numerator, new value is updated only with rising edge of Execute input
	RatioDenominator	INT	Gear ratio denominator, new value is updated only with rising edge of Execute input
	MasterValueSource	MC_Source	Decides to use the actual position or reference position of master axis. - mcSetValue - Synchronization on master set value. - mcActualValue - Synchronization on master actual value
	MasterSyncPosition	LREAL	The position of the master where the slave is insync with the master
	SlaveSyncPosition	LREAL	Slave Position at which the axes are running in sync
	SyncDirection	MC_Direction	Moving direction for the slave to start the movement. Applicable: POSITIVE or NEGATIVE, use SHORTEST for any other value
	SyncWindow	LREAL	[u], Used to determine the moving direction, combined with SyncDirection - When the slave is outside the SyncWindow, it will move the direction which is given in SyncDirection - When the slave is inside the SyncWindow, it will move SHORTEST to meet the SlaveSyncPosition
	Velocity	LREAL	[u/s], Velocity which limits the synchronization movement. The slave has to be able to move faster than the master axis, otherwise it is possible the SlaveSyncPosition is never reached when the master starts to move. Range: >0
	Acceleration	LREAL	[u/s ²], Acceleration which limits the synchronization movement. Range: >0
	BufferMode	MC_BufferMode	Not supported, default mcABORTING used
Output	StartSync	BOOL	Synchronization was started
	InSync	BOOL	Commanded gearing completed
	Busy	BOOL	The function block is not finished
	Active	BOOL	Indicates that the function block has control on the axis
	CommandAborted	BOOL	Command is aborted by another command from other PLCopen function block
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. Refer to Enumeration ERROR_ID
Inout	Master	Axis_Ref	Reference to master axis
	Slave	Axis_Ref	Reference to slave axis

10.2.1.1.1.7 MCA_Indexing (FB)

This function block will, upon a rising edge on Execute, do a number of relative or absolute moves, listed in a table (Array of MCA_Pos_Ref).



The function block will position the axis to a complete stop at target position and continue with the next move from the table automatically.



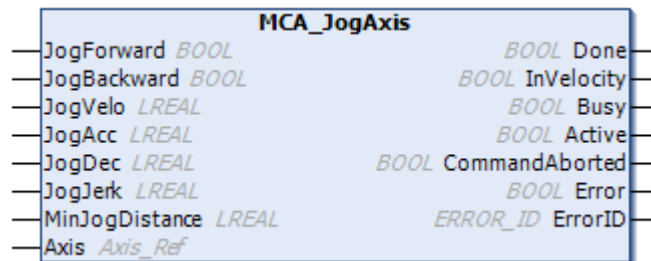
InOut:

Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
	pPositions	POINTER TO MCA_Pos_ Ref	POS_REF Reference to Structure or array with relative move distances listed. Typical type of Positions: LREAL, The array needs to have at least (TableIndex + MovesToDo-1) elements
	MovesToDo	WORD	Provide number of moves, not more than entries in Table
	TableIndex	WORD	Index to an array of MCA_Pos_Ref, points to the movement to be performed on rising edge of Execute, start with 1 for the first entry
Out-put	Done	BOOL	Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BOOL	The function block is not finished
	Active	BOOL	Indicates that the function block has control on the axis
	Command-Aborted	BOOL	Command is aborted by another command from other PLCopen function block
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
	MovesPending	WORD	Indicates the number of moves still to execute
	IndexNo	WORD	Index executing or last index completed, starting with "1"
Inout	Axis	Axis_Ref	Reference to axis

10.2.1.1.1.8

10.2.1.1.1.9 MCA_JogAxis (FB)

This function block jogs an axis for at least a given distance forward or backward with the selected jog velocity and acceleration.



This function block, after rising edge on JogForward or JogBackward, start a continuous move (at least for the minimum distance) and continue upon high-level on these inputs with a continuous motion, until they are FALSE, then on their falling edge, the axis is regularly decelerated to stop. The movement is carried out on Jog velocity for the minimum distance or longer.

- In case of both enable signals (JogForward and JogBackward) are high, the function block will not move the axis.
- In case of MinJogDistance = 0, no specified distance is moved and the movement will stop as soon as the JogForward and JogBackward is FALSE.

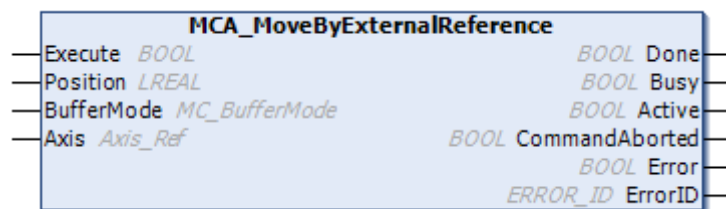
InOut:

Scope	Name	Type	Comment
Input	JogForward	BOOL	Move forward as long as JogForward = TRUE or at least MinJogDistance
	JogBackward	BOOL	Move backward as long as JogForward = TRUE or at least MinJogDistance
	JogVelo	LREAL	[u/s] Velocity to jog. Range: >0. If value = 0, JogVelo will be equal to parameter paraMaxVelocityAppl
	JogAcc	LREAL	[u/s ²] Acceleration to jog. Range: >0. If value = 0, JogAcc will be equal to parameter paraMaxAccelerationAppl
	JogDec	LREAL	[u/s ²] Deceleration to jog. Range: >0. If value = 0, JogDec will be equal to parameter paraMaxDecelerationAppl
	JogJerk	LREAL	[u/s ³] Jerk value for jog. Range: >=0, <0: value at input ignored
	MinJog-Distance	LREAL	[u] Minimum distance to jog

Scope	Name	Type	Comment
Output	Done	BOOL	Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BOOL	The function block is not finished
	Active	BOOL	Indicates that the function block has control on the axis
	Command-Aborted	BOOL	Command is aborted by another command from other PLCopen function block
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ER-ROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
Inout	Axis	Axis_Ref	Reference to axis

10.2.1.1.1.10 MCA_MoveByExternalReference (FB)

This function block gives a reference position to the axis which is directly passed to the position control loop.



The axis will follow the given position without a ramp but immediately. The reference position is evaluated continuously.

To stop the motion, the function block has to be interrupted by another function block issuing a new command



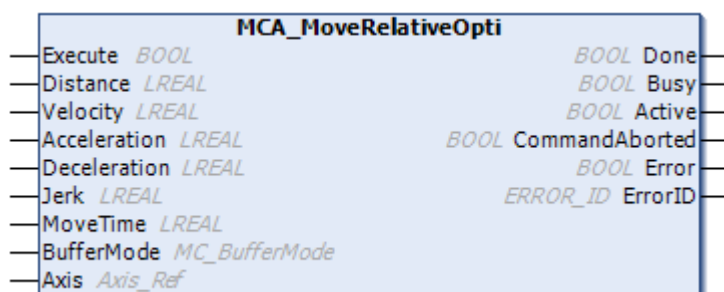
This block has to be called within the REAL-TIME task, same task as CMC_Basic_Kernel

InOut:

Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
	Position	LREAL	[u] Reference position. New values are accepted in a running system without a new rising edge of Execute
	Buffer-Mode	MC_BufferMode	Not supported, default mcABORTING used
Out-put	Done	BOOL	Shows the status of the function block. This output not used in current version
	Busy	BOOL	The function block is not finished
	Active	BOOL	Indicates that the function block has control on the axis
	Command-Aborted	BOOL	Command is aborted by another command from other PLCopen function block
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
Inout	Axis	Axis_Ref	Reference to axis

10.2.1.1.11 MCA_MoveRelativeOpti (FB)

This function block commands a controlled motion of a specified distance relative to the actual position at the time of the execution.



This function block is designed to allow an easier setup for positioning movement. The input MoveTime holds the allowed time to move the given distance. The other inputs: Velocity, Acceleration, Deceleration and Jerk can be left "0", then the movement will use the system limits and perform as "soft" as possible positioning.

- It will always use a Jerk (when possible).
- To switch off the usage of Jerk, use Jerk = -1 as input parameter.
- Use the smallest possible acceleration/deceleration.
- Use the smallest possible velocity.

If it is not required to create a movement with limited Jerk, the input Jerk = -1 should be used. The acceleration and deceleration will be applied with a step but a smaller maximum value is required.

If parameters like Velocity, Acceleration and Deceleration are set, these will be considered to be the upper limits during the movement. If it is not possible to execute the movement in the given time, an error(3 - PARAMETER_EXCEEDS_LIMIT) will be shown. The function block also gives a suggestion which values could be used to execute the movement within the time limitations.



If parameters are given which are not possible to use, e.g. velocity is too small to reach the target in the given time, the function block's internal variables usedJerk, usedVelocity, usedAcceleration and usedDeceleration will show a possible solution.

- If the velocity was too small, the internal variable usedVelocity will hold the smallest possible value to execute the movement in the given time. This would mean execute it without any ramps. Internal variables usedAcceleration and usedDeceleration will be "0" in this case.
- If the velocity was ok, but acceleration and deceleration are too small, usedVelocity will show the value from input Velocity and variables usedAcceleration and usedDeceleration will hold the necessary values to execute the movement in the given time.



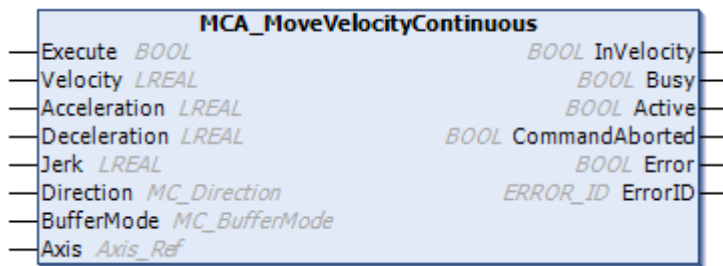
With "MoveTime = 0", the behavior for the function block is identical with MC_MoveRelative. It is possible to use just Distance and MoveTime as input parameters. In this case, the function block will take the axis configuration parameters as limitations for the movement and always create the smoothest possible interpolation.

InOut:

Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
	Distance	LREAL	[u] = Technical unit, Relative distance for the motion
	Velocity	LREAL	[u/s] Value of the maximum velocity (not necessarily reached). Range: >0
	Acceleration	LREAL	[u/s ²] Value of the acceleration (increasing energy of the motor). Range: >0
	Deceleration	LREAL	[u/s ²] Value of the deceleration (decreasing energy of the motor). Range: >0
	Jerk	LREAL	[u/s ³] Value of the jerk. Range: >=0. To switch off the usage of Jerk, use Jerk = -1
	MoveTime	LREAL	[s] Allowed time to complete the movement. Range: >=0
	Buffer-Mode	MC_BufferMode	Not supported, default mcABORTING used
Out-put	Done	BOOL	Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BOOL	The function block is not finished
	Active	BOOL	Indicates that the function block has control on the axis
	Command-Aborted	BOOL	Command is aborted by another command from other PLCopen function block
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
Inout	Axis	Axis_Ref	Reference to axis

10.2.1.1.1.12 MCA_MoveVelocityContinuous (FB)

This function block commands a never ending controlled motion at a specified velocity.



The difference to function block MC_MoveVelocity is that the values for Velocity, Acceleration and Deceleration can be modified continuously. If there is a change of the velocity, the reaction on the signal InVelocity will be delayed for 1 cycle.

- To stop the motion, the function block has to be interrupted by another function block issuing a new command.
- The signal “InVelocity” is set when the commanded velocity equals the velocity input.
- Velocity, Acceleration, Deceleration and Jerk might be changed and will be used continuously
- In combination with MC_MoveSuperimposed, the output “InVelocity” stays TRUE once the velocity setpoint of the axis has reached the commanded velocity

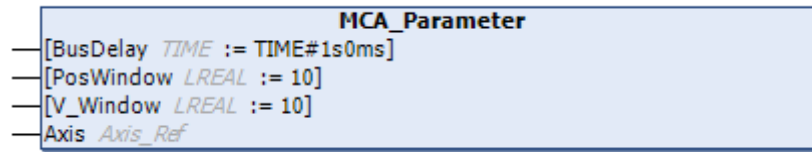
InOut:

Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
	Velocity	LREAL	[u/s] Value of the maximum velocity (not necessarily reached). Range: >0
	Acceleration	LREAL	[u/s ²] Value of the acceleration (increasing energy of the motor). Range: >0. If value = 0, Acceleration will be equal to parameter paraMaxAccelerationAppl
	Deceleration	LREAL	[u/s ²] Value of the deceleration (decreasing energy of the motor). Range: >0. If value = 0, Deceleration will be equal to parameter paraMaxDecelerationAppl
	Jerk	LREAL	[u/s ³] Value of the jerk. Range: >=0
	Direction	MC_Direction	Positive, Negative, otherwise use Current as default
	BufferMode	MC_BufferMode	Not supported, default mcABORTING used
Output	InVelocity	BOOL	Commanded velocity is reached
	Busy	BOOL	The function block is not finished
	Active	BOOL	Indicates that the function block has control on the axis

Scope	Name	Type	Comment
	CommandAborted	BOOL	Command is aborted by another command from other PLCOpen function block
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
Inout	Axis	Axis_Ref	Reference to axis

10.2.1.1.1.13 MCA_Parameter (FB)

This function block can be used to change the default values of the following parameters:



- Target position window. The default value is 10 units.
- Target velocity window. The default value is 10 units.
- Maximum fieldbus delay. If this value will be exceeded then it will be assumed that there is a communication error



The block MCA_Parameter has to be used to adjust the velocity limit when velocities < 10 u/s should be used (10 u/s: default value).

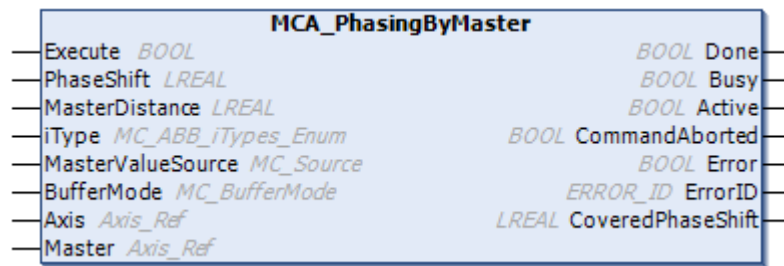
- The parameter v_Window defines the limit for the axis to reach its target velocity or standstill
- This detection will not work properly when smaller velocities are used, especially the block MC_StepLimitSwitch will not stop the axis when reaching the switch!

InOut:



Scope	Name	Type	Initial	Comment
Input	BusDelay	TIME	TIME#1s0ms	A delay time to wait for fieldbus data. When the delay time is too long, the reaction time of function blocks might be increased, while when it is too short an error might be indicated although everything is ok
	PosWindow	LREAL	10	The limit for the axis to reach it's target position
	V_Window	LREAL	10	The limit for the axis to reach it's target velocity
Inout	Axis	Axis_Ref		Reference to axis

10.2.1.1.14 MCA_PhasingByMaster (FB)

This function block performs a movement for the relation to the master axis of the specified axis. A real movement is just performed in case the axis is in synchronized motion.



This function block creates a relative phase shift in the master position of a slave axis. The master position is shifted in relation to the real physical position. This is analogous to opening a coupling on the master shaft for a moment, and is used to delay or advance an axis to its master. The phase shift is seen from the slave. The master does not know that there is a phase shift experienced by the slave. The phase shift remains until another "Phasing" command changes it again.

	The phasing is executed with respect to a master movement and will use a polynomial interpolation
	To halt this function block user must use MC_HaltPhasing function block instead of MC_Stop or MC_Halt.

InOut:

Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
	PhaseShift	LREAL	[u] = Technical unit, phase difference in master
	MasterDistance	LREAL	[u] Distance master has to move
	iType	MC_ABB_i-Types_Enum	Interpolation type, MCA_LINEAR, MCA_POLY3 or MCA_POLY5 are applicable
	MasterValueSource	MC_Source	Decide to use the actual position or reference position of master axis
	BufferMode	MC_Buffer-Mode	Not supported, default mcABORTING used

Scope	Name	Type	Comment
Output	Done	BOOL	Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BOOL	The function block is not finished
	Active	BOOL	Indicates that the function block has control on the axis
	CommandAborted	BOOL	Command is aborted by another command from other PLCopen function block
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
	CoveredPhaseShift	LREAL	Actual phase shift of master axis to slave axis, valid while function block is busy
Inout	Axis	Axis_Ref	Reference to axis
	Master	Axis_Ref	Reference to master axis

10.2.1.1.15 MCA_ReadParameterList (FB)

The function block reads a list of parameters by using the “MC_ReadParameter”.



The rules for utilizing the function block correspond to MC_ReadParameter as well as the ErrorIDs. All parameters and parameter numbers have to be stored in an array of type MCA_Parameter_Struct. The address of the first element is given to the function blocks input “pParameters”. The number of elements to be read is declared at input “Num”.

No (PN)	Parameter Name	Data-Type	Min.	Max.	Default	R/W	Comments
1	Commanded-Position	DINT				R	Commanded Position
2	SWLimitPos	DINT	-2147483647	2147483647	2147483647	R/W	Positive Software limit switch position
3	SWLimitNeg	DINT	-2147483647	2147483647	-2147483647	R/W	Negative Software limit switch position
4	EnableLimitPos	BOOL	FALSE	TRUE	FALSE	R/W	Enable positive software limit switch
5	EnableLimit-Neg	BOOL	FALSE	TRUE	FALSE	R/W	Enable negative software limit switch
6	EnablePostLag-Monitoring	BOOL	FALSE	TRUE	TRUE	R/W	Enable monitoring of position lag (following error)
7	MaxPosition-Lag	DINT	1	2147483647***		R	Maximal position lag
8	MaxVelocitySystem	DINT			32767	R	Maximal allowed velocity of the axis in the motion system
9	MaxVelocityAppl	DINT	0**	32767	32767	R/W	Maximal allowed velocity of the axis in the application
10	ActualVelocity	DINT	-32767	32767		R	Actual velocity
11	Commanded-Velocity	DINT	-32767	32767		R	Commanded Velocity
12	MaxAccelerationSystem	DINT			32767	R	Maximal allowed acceleration of the axis in the motion system

No (PN)	Parameter Name	Data-Type	Min.	Max.	Default	R/W	Comments
13	MaxAccelerationAppl	DINT	10	32767	32767	R/W	Maximal allowed acceleration of the axis in the application
14	MaxDecelerationSystem	DINT	w		32767	R	Maximal allowed deceleration of the axis
15	MaxDecelerationAppl	DINT	10	32767	32767	R/W	Maximal allowed deceleration of the axis
16	MaxJerk	DINT	0*	2147483647	2147483647	R/W	Maximal allowed jerk of the axis
2001	MODULO_NO-MINATOR	DINT	1	2147483647	1	R/W	ABB specific parameter. Used for Central Motion Control implementation: Gearbox modifier to MODULO_RANGE
2002	MODULO_DENOMINATOR	DINT	1	2147483647	1	R/W	ABB specific parameter. Used for Central Motion Control implementation: Gearbox modifier to MODULO_RANGE
2003	EnableLimit2Decelerate	BOOL	FALSE	TRUE	FALSE	R/W	Enable software limit switches to decelerate
2004	EnableLimitAbort	BOOL	FALSE	TRUE	FALSE	R/W	Enable software limit switches to abort ongoing movement
2005	EnableLimitVelocity	BOOL	FALSE	TRUE	FALSE	R/W	If the velocity is limited the unmoved position will be covered whenever possible
2006	SWLimit2DecPos	LREAL	-2147483647	2147483647	2147483647	R/W	Used as end position for EnableLimit2Decelerate
2007	SWLimit2DecNeg	LREAL	-2147483647	2147483647	2147483647	R/W	Used as end position for EnableLimit2Decelerate
2008	MaxPositionGapLL	LREAL	0	214748364700	0	R/W	Used to stop the ongoing movement if position is behind

*0 means: no limitation of jerk is performed.

**Axis will stay in stop.

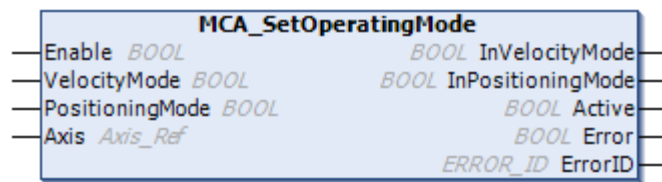
***Is modified by CMC_Axis_Control_Parameter, the maximum Value is calculated in increments, the value which is delivered by ReadParameter will be given in [u].

InOut:

Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
	Num	WORD	Number of parameters to read
	pParameters	POINTER TO MCA_Parameter_Struct	Points to an array of type MCA_Parameter_Struct which holds the parameter numbers and values
Output	Done	BOOL	Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BOOL	The function block is not finished
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
Inout	Axis	Axis_Ref	Reference to axis

10.2.1.1.1.16 MCA_SetOperatingMode (FB)

This function block changes the axis mode from positioning to velocity mode and vice versa.



By default, an axis is always a positioning axis which has to follow either the drives or the PLCs position control loop. In some applications, the movement is limited (e.g. by torque restrictions) so the position can't be reached. A position controlled axis would first speed up and then create a following error, both caused by the increasing position lag.

The function block MCA_SetOperatingMode can be used to prevent this behavior and will switch between velocity and position controlled behavior "on the fly".

In velocity mode

- The Speed_Reference is created by feed-forward, in an open loop. (It is not required to set FF_Percentage parameter)
- Reference_Position will follow the Drive_Actual_Position
- Position following error is not supervised.
- It will be immediately effective in axis state: STANDSTILL, DISABLED or ERRORSTOP
- In any other mode, it will be effective with the next "Execute" rising edge on a function block which activates a movement. It can be called while the axis is moving and will create a bumpless transition between the velocity- and position controlled mode



To use the function block MCA_SetOperatingMode, the drive has to be used in CSV (ContinuousSynchronousVelocity), or an analog drive has to be used, which means it has to move controlled by Speed_Reference (Kernel).

Precondition: The axis has to be used with position loop controlled by the PLC, otherwise a velocity mode is not possible

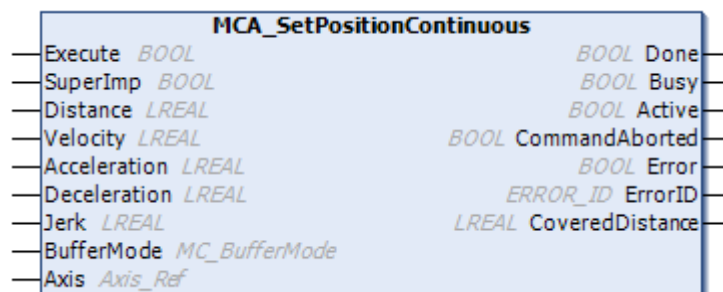
- It will not do any mode changes if both inputs = FALSE
- It will not do any mode changes if both inputs = TRUE

InOut:

Scope	Name	Type	Comment
Input	Enable	BOOL	Enables the function block to switch the operating mode. A rising edge is not required, the block will be activated if Enable = TRUE and will react to the VelocityMode/PositioningMode inputs
	VelocityMode	BOOL	VelocityMode = TRUE/PositioningMode = FALSE: Switch axis to velocity mode
	Positioning-Mode	BOOL	VelocityMode = FALSE/PositioningMode = TRUE: Switch axis to positioning mode
Output	InVelocityMode	BOOL	Shows the axis state is in Velocity Mode
	InPositioning-Mode	BOOL	Shows the axis state is in Positioning Mode
	Active	BOOL	Indicates that the selected mode is activated
	Error	BOOL	Signals that an error has occurred within the function block, in this function block no error is generated
	ErrorID	ERROR_ID	Error identification, in this function block no error is generated
Inout	Axis	Axis_Ref	Reference to axis

10.2.1.1.17 MCA_SetPositionContinuous (FB)

This function block modifies the position of an axis with a defined profile.



This function block shifts the coordinate system of an axis by manipulating either the setpoint position or the actual position of an axis. This can be used for instance for a reference situation “on the fly” where no abrupt position change is allowed, example when a slave axis is linked to the modified axis. This function block can also be used during motion without changing the commanded position, which is now positioned in the

shifted coordinate system. A continuous position correction will be achieved, with a defined profile.

- The function block is allowed in any state except ErrorStop or Homing. In Discrete Motion, just mode SuperImp = TRUE is possible.
- The block will not change the axis state even when it results in a movement.
- With SuperImp = TRUE, the axis will hold the setpoint position while an offset is applied to the actual position. This will result in a movement as the position control loop will keep the distance between setpoint- and actual position constant. A slave axis will not see this movement and will not follow. When the block is ready, the axis will have moved physically by -Distance but the positions in Axis_Ref will not have been changed.
- With SuperImp = FALSE, the behavior equals MC_SetPosition, but executed continuously. The axis will physically stay where it is but the actual position and setpoint position are modified. A slave axis will follow.
- With SuperImp = FALSE: When it is acceptable and required to correct the position with a jump, use Acceleration = -1.



The result will be lost when using MC_SetPosition, execute a Homing, or set Enable = FALSE for CMC_Basic_Kernel

InOut:

Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
	SuperImp	BOOL	Defines 2 different modes. TRUE= Superimposed movement
	Distance	LREAL	[u] = Technical unit, Relative distance for the motion
	Velocity	LREAL	[u/s] Value of the maximum velocity (not necessarily reached). Range: >0
	Acceleration	LREAL	[u/s ²] Value of the acceleration (increasing energy of the motor). Range: >0
	Deceleration	LREAL	[u/s ²] Value of the deceleration (decreasing energy of the motor). Range: >0
	Jerk	LREAL	[u/s ³] Value of the jerk. Range: >=0
	BufferMode	MC_Buffer-Mode	Not supported, default mcABORTING used

Scope	Name	Type	Comment
Out-put	Done	BOOL	Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BOOL	The function block is not finished
	Active	BOOL	Indicates that the function block has control on the axis
	CommandAborted	BOOL	Command is aborted by another command from other PLCopen function block
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
	CoveredDistance	LREAL	Shows the progress, starts with 0 and ends with CoveredDistance = Distance. The value is valid while the function block is active
Inout	Axis	Axis_Ref	Reference to axis

10.2.1.1.18 MCA_WriteParameterList (FB)

The function block writes a list of parameters by using the “MC_WriteParameter”.



The rules for utilizing the function block correspond to MC_WriteParameter as well as the ErrorIDs. All parameters and parameter numbers have to be stored in an array of type MCA_Parameter_Struct. The address of the first element is given to the function blocks input “pParameters”. The number of elements to be written is declared at input “Num”.

No	Parameter Name	Data-Type	Min.	Max.	Default	R/W	Comments
1	Commanded-Position	DINT				R	Commanded Position
2	SWLimitPos	DINT	-2147483647	2147483647	2147483647	R/W	Positive Software limit switch position
3	SWLimitNeg	DINT	-2147483647	2147483647	-2147483647	R/W	Negative Software limit switch position
4	EnableLimitPos	BOOL	FALSE	TRUE	FALSE	R/W	Enable positive software limit switch
5	EnableLimitNeg	BOOL	FALSE	TRUE	FALSE	R/W	Enable negative software limit switch
6	EnablePostLag-Monitoring	BOOL	FALSE	TRUE	TRUE	R/W	Enable monitoring of position lag (following error)
7	MaxPosition-Lag	DINT	1	2147483647***		R	Maximal position lag
8	MaxVelocitySystem	DINT			32767	R	Maximal allowed velocity of the axis in the motion system
9	MaxVelocityAppl	DINT	0**	32767	32767	R/W	Maximal allowed velocity of the axis in the application
10	ActualVelocity	DINT	-32767	32767		R	Actual velocity
11	Commanded-Velocity	DINT	-32767	32767		R	Commanded Velocity
12	MaxAccelerationSystem	DINT			32767	R	Maximal allowed acceleration of the axis in the motion system

13	MaxAccelerationAppl	DINT	10	32767	32767	R/W	Maximal allowed acceleration of the axis in the application
14	MaxDecelerationSystem	DINT			32767	R	Maximal allowed deceleration of the axis
15	MaxDecelerationAppl	DINT	10	32767	32767	R/W	Maximal allowed deceleration of the axis
16	MaxJerk	DINT	0*	2147483647	2147483647	R/W	Maximal allowed jerk of the axis
2001	MODULO_NO-MINATOR	DINT	1	2147483647	1	R/W	ABB specific parameter. Used for Central Motion Control implementation: Gearbox modifier to MODULO_RANGE
2002	MODULO_DE-NOMINATOR	DINT	1	2147483647	1	R/W	ABB specific parameter. Used for Central Motion Control implementation: Gearbox modifier to MODULO_RANGE
2003	EnableLimit2Decelerate	BOOL	FALSE	TRUE	FALSE	R/W	Enable software limit switches to decelerate
2004	EnableLimitAbort	BOOL	FALSE	TRUE	FALSE	R/W	Enable that software limit switches will abort ongoing movement
2005	EnableLimitVelocity	BOOL	FALSE	TRUE	FALSE	R/W	If the velocity is limited the unmoved position will be covered whenever possible
2006	SWLimit2DecPos	LREAL	-2147483647	2147483647	2147483647	R/W	Used as end position for EnableLimit2Decelerate
2007	SWLimit2DecNeg	LREAL	-2147483647	2147483647	2147483647	R/W	Used as end position for EnableLimit2Decelerate
2008	MaxPositionGapLL	LREAL	0	214748364700	0	R/W	Used to stop the ongoing movement if position is behind

*0 means: no limitation of jerk is performed.

**Axis will stay in stop.

***Is modified by CMC_Axis_Control_Parameter, the maximum Value is calculated in increments, the value which is delivered by ReadParameter will be given in [u].

InOut:

Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
	Num	WORD	Number of parameters to write
	pParameters	POINTER TO MCA_Parameter_Struct	Points to an array of type MCA_Parameter_Struct which holds the parameter numbers and values
Out-put	Done	BOOL	Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BOOL	The function block is not finished
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
Inout	Axis	Axis_Ref	Reference to axis

10.2.1.1.1.19 MCA_DigitalCamSwitch (FB)

This function block implements Tappet functionality. The output pin is Switched on and Off based on TrackID, Axis Position and Configuration in the MCA_CAMTappet array.

Users can either create MCA_CAMTappet array manually or by using the CamEditor to configure tappets.

When using the CamEditor, MCA_CAMTappet array is generated automatically based on the configuration and the array name follows the Cam object name with “_T”.

For example, Cam object’s name is “RotaryShear”, MCA_CAMTappet array is generated as “RotaryShear_T”. This can be directly passed to input pin name “Tappets” of function block MCA_Tappet



User can utilize CAM editor in Automation builder to generate Tappet Table (MCA_CAMTappet) automatically. AxisPosition in MCA_CAMTappet should be always in ascending order.

InOut:

Scope	Name	Type	Comment
Input	Enable	BOOL	Enables the function block to evaluate tappet track. A rising edge is not required, the block will be activated if Enable = TRUE
	TrackID	INT	Track ID for which Output is needed. Range: >=1
Out-put	InOperation	BOOL	True indicates that Function Block is in Operation
	Output	BOOL	Actual tappet value as per tappet table
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
Input	Tappets	POINTER TO MCA_CAMTappet	Reference to CAM_T description, points to an array of MCA_CAMTappet, Must be an Array not a pointer
Inout	AxisRef	Axis_Ref	Axis Ref: Reference to Tappet Axis.

10.2.1.1.1.20 MCA_MoveBuffer (FB)

This function block will, upon a rising edge on Execute, do a number of relative or absolute moves, listed in a table (Array of MCA_Pos_Ref).

With VelocityMode = FALSE, the function block will position the axis to a complete stop at target position and continue with the next move from the table automatically.

With VelocityMode = TRUE, the function block will end the positioning movement with the given velocity and (if it is not the last movement) continue with the next move from the table automatically.

If it is the last movement, the axis will come to a complete stop.

InOut:

Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
	Move-sToDo	WORD	Provide number of moves, not more then entries in MoveParameter
	TableIndex	WORD	Index to an array of MCA_Pos_Ref, points to the movement to be performed on rising edge of Execute, start with 1 for the first entry
Output	Done	BOOL	Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BOOL	The function block is not finished
	Active	BOOL	Indicates that the function block has control on the axis
	CommandAborted	BOOL	Command is aborted by another command from other PLCopen function block
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
	Move-sPending	WORD	Indicates the number of moves still to execute
	IndexNo	WORD	Index executing or last index completed, starting with "1"
Input	MoveParameter	POINTER TO MCA_POS_REF	array with all parameters for either absolute or relative positioning movement
Inout	Axis	Axis_Ref	Reference to axis

10.2.1.1.1.21 MCA_MoveByExtRefRelative (FB)

This function block gives a reference position to the axis which is directly passed to the position control loop. The positioning is relative, starting with a rising edge at Execute.

The axis will follow the given position without a ramp but immediately. The reference position is evaluated continuously.

To stop the motion, the function block has to be interrupted by another function block issuing a new command



This block has to be called within the REAL-TIME task, same task as CMC_Basic_Kernel

InOut:

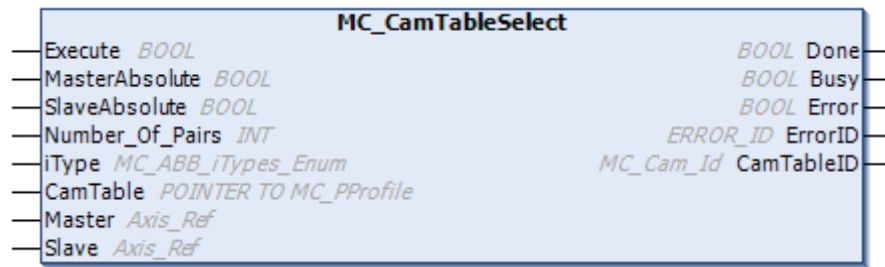
Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
	Distance	LREAL	[u] Reference position, as relative position. New values are accepted in a running system without a new rising edge of Execute. The movement is relative from the 1. rising edge. Distance is not allowed to exceed +/-0x7FFFFFFF. The value will be limited accordingly.
	Buffer-Mode	MC_BufferMode	Not supported, default mcABORTING used
Output	Done	BOOL	Shows the status of the function block. This output not used in current version
	Busy	BOOL	The function block is not finished
	Active	BOOL	Indicates that the function block has control on the axis
	CommandAborted	BOOL	Command is aborted by another command from other PLCopen function block
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
Inout	Axis	Axis_Ref	Reference to axis

10.2.1.1.2 MC Administrative

PLC open motion control Administrative function blocks

10.2.1.1.2.1 MC_CamTableSelect (FB)

This function block selects the CAM tables by setting the connections to the relevant tables



- A virtual axis can be used as master axis.
- MC_PProfile is an ABB specific data type.
- CamTableSelect makes data available. It prepares the provided data to be used by MC_CamIn or MCA_CamInDirect

Example on adding a CAM table:

```
CamTable:ARRAY[0..2]OF MC_PProfile := [
(Mastr_position:= 0, interpolation_point:= 0, Velocity_ratio:= 1, Acceleration_ratio:= 1),
(Master_position:= 10, interpolation_point:= 10, Velocity_ratio:= 1, Acceleration_ratio:= 1
), (Master_position:= 20, interpolation_point:= 15, Velocity_ratio:= 1, Acceleration_ratio:=
1),
];
```



User can utilize CAM editor in Automation builder to generate Cam table (MC_PProfile) automatically. For more details refer to Automation builder help.

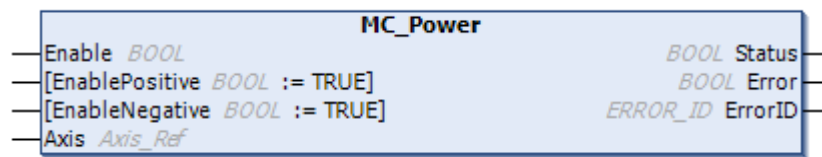
InOut:

Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
	MasterAbsolute	BOOL	CamTable holds absolute positions for master. TRUE = Absolute, FALSE = Relative coordinates
	SlaveAbsolute	BOOL	CamTable holds absolute positions for slave. TRUE = Absolute, FALSE = Relative coordinates
	Number_Of_Pairs	INT	Number of sampling points, number of elements in CamTable
	iType	MC_ABB_iTypes_Enum	Type of interpolation. Possible values are: MCA_SPLINE_COMPLETE MCA_SPLINE_NATURAL

Scope	Name	Type	Comment
			MCA_POLY5 MCA_POLY3 MCA_LINEAR
	CamTable	POINTER TO MC_PProfile	Reference to CAM description, points to an array of MC_PProfile
Output	Done	BOOL	Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BOOL	The function block with Busy = TRUE has control on the axis
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
	CamTableID	MC_Cam_Id	Refers to prepared data, to be used for MC_CamIn or MCA_CamInDirect
Inout	Master	Axis_Ref	Reference to master axis
	Slave	Axis_Ref	Reference to slave axis

10.2.1.1.2.2 MC_Power (FB)

This function block controls the power stage (on or off).



If this function block is called with Enable = TRUE and axis Disabled, it will either lead to StandStill when no error inside the axis or ErrorStop if an error exists. An error indicates that there is a hardware problem with the power stage.

- If power fails (also during operation) block will generate a transition to the state ErrorStop.
- When MC_Power is called with Enable = FALSE the axis goes to state Disabled for every state including ErrorStop.
- It is possible to set an error variable when the Command is TRUE for a while and the Status remains FALSE with a Timer Function Block and an AND Function (with inverted Status input). It indicates that there is a hardware problem with the power stage



Error "NO_FIELD_ACCESS" can get generated at the output ErrorID even when function block is disabled.

InOut:

Scope	Name	Type	Initial	Comment
Input	Enable	BOOL		Enable continuously on "TRUE"
	EnablePositive	BOOL	TRUE	Enable positive movement direction, FALSE will hold the axis without ramp, an ongoing movement is aborted
	EnableNegative	BOOL	TRUE	Enable negative movement direction, FALSE will hold the axis without stop, an ongoing movement is aborted
Output	Status	BOOL		Axis is activated
	Error	BOOL		Signals that error has occurred within function block
	ErrorID	ERROR_ID		Error identification. For error details refer to Enumeration ERROR_ID
Inout	Axis	Axis_Ref		Reference to axis

10.2.1.1.2.3 MC_ReadActualPosition (FB)

This function block returns the actual position.

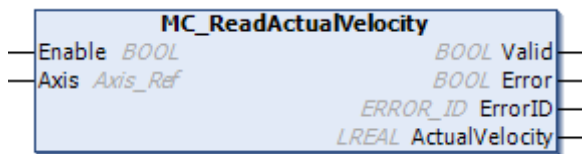


InOut:

Scope	Name	Type	Comment
Input	Enable	BOOL	Read continuously on “TRUE”
Output	Valid	BOOL	Value is available
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ER-ROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
	Position	LREAL	[u] New absolute position
Inout	Axis	Axis_Ref	Reference to axis

10.2.1.1.2.4 MC_ReadActualVelocity (FB)

This function block returns the value of the actual velocity as long as Enable is set.



Output Valid is true when the data-output “Velocity” is valid. If Enable is reset, the data loses its validity, and all outputs are reset, no matter if new data is available.



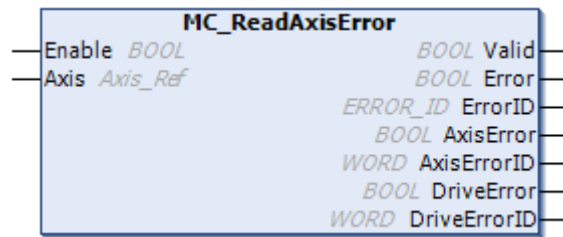
The output ActualVelocity is a signed value.

InOut:

Scope	Name	Type	Comment
Input	Enable	BOOL	Read continuously on “TRUE”
Output	Valid	BOOL	Value is available
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ER-ROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
	ActualVelocity	LREAL	The value of Actual velocity
Inout	Axis	Axis_Ref	Reference to axis

10.2.1.1.2.5 MC_ReadAxisError (FB)

This function block describes general axis errors not relating to the PLCopen function blocks.



The error codes are generated by CMC_Basic_Kernel



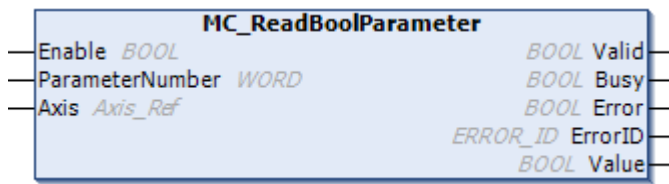
This function block is the equivalent to read the AxisErrorID parameter using MC_ReadParameter

InOut:

Scope	Name	Type	Comment
Input	Enable	BOOL	Read continuously on "TRUE"
Out-put	Valid	BOOL	Value is available
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ER-ROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
	AxisError-ID	WORD	Axis error. It shows the CMC_Basic_Kernel ErrorID, Refer to enumeration ERROR_ID
Inout	Axis	Axis_Ref	Reference to axis

10.2.1.1.2.6 MC_ReadBoolParameter (FB)

It will support the Parameters defined as “standard” in the PLCopen. These parameters will be transferred to the units as used for PLCopen function blocks



When a drive based axis implementation is used, the function block returns the value of a drive specific BOOL parameter or a PLCopen BOOL parameter. When the Central Motion Control axis implementation is used, a BOOL parameter according to the list PLCopen parameter is returned

No	Parameter Name	Data Type	Min.	Max.	Default	R/W	Comments
1	Commanded-Position	DINT				R	Commanded Position
2	SWLimitPos	DINT	-2147483647	2147483647	2147483647	R/W	Positive Software limit switch position
3	SWLimitNeg	DINT	-2147483647	2147483647	-2147483647	R/W	Negative Software limit switch position
4	EnableLimitPos	BOOL	FALSE	TRUE	FALSE	R/W	Enable positive software limit switch
5	EnableLimit-Neg	BOOL	FALSE	TRUE	FALSE	R/W	Enable negative software limit switch
6	EnablePostLag-Monitoring	BOOL	FALSE	TRUE	TRUE	R/W	Enable monitoring of position lag (following error)
7	MaxPosition-Lag	DINT	1	2147483647***		R	Maximal position lag
8	MaxVelocitySystem	DINT			32767	R	Maximal allowed velocity of the axis in the motion system
9	MaxVelocityAppl	DINT	0**	32767	32767	R/W	Maximal allowed velocity of the axis in the application
10	ActualVelocity	DINT	-32767	32767		R	Actual velocity
11	Commanded-Velocity	DINT	-32767	32767		R	Commanded Velocity

No	Parameter Name	Data Type	Min.	Max.	Default	R/W	Comments
12	MaxAccelerationSystem	DINT			32767	R	Maximal allowed acceleration of the axis in the motion system
13	MaxAccelerationAppl	DINT	10	32767	32767	R/W	Maximal allowed acceleration of the axis in the application
14	MaxDecelerationSystem	DINT			32767	R	Maximal allowed deceleration of the axis
15	MaxDecelerationAppl	DINT	10	32767	32767	R/W	Maximal allowed deceleration of the axis
16	MaxJerk	DINT	0*	2147483647	2147483647	R/W	Maximal allowed jerk of the axis
2001	MODULO_NO-MINATOR	DINT	1	2147483647	1	R/W	ABB specific parameter. Used for Central Motion Control implementation: Gearbox modifier to MODULO_RANGE
2002	MODULO_DE-NOMINATOR	DINT	1	2147483647	1	R/W	ABB specific parameter. Used for Central Motion Control implementation: Gearbox modifier to MODULO_RANGE
2003	EnableLimit2Decelerate	BOOL	FALSE	TRUE	FALSE	R/W	Enable software limit switches to decelerate
2004	EnableLimitAbort	BOOL	FALSE	TRUE	FALSE	R/W	Enable that software limit switches will abort ongoing movement
2005	EnableLimitVelocity	BOOL	FALSE	TRUE	FALSE	R/W	If the velocity is limited the unmoved position will be covered whenever possible

No	Parameter Name	Data Type	Min.	Max.	Default	R/W	Comments
2006	SWLimit2DecPos	LREAL	-2147483647	2147483647	2147483647	R/W	Used as end position for EnableLimit2Decelerate
2007	SWLimit2DecNeg	LREAL	-2147483647	2147483647	2147483647	R/W	Used as end position for EnableLimit2Decelerate
2008	MaxPosition-GapLL	LREAL	0	214748364700	0	R/W	Used to stop the ongoing movement if position is behind

*0 means: no limitation of jerk is performed.

**Axis will stay in stop.

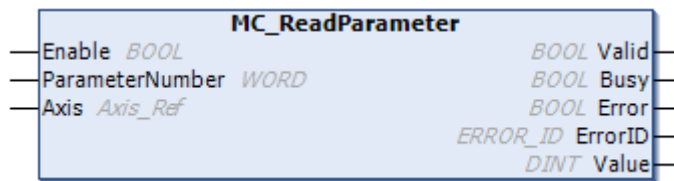
***Is modified by CMC_Axis_Control_Parameter, the maximum Value is calculated in increments, the value which is delivered by ReadParameter will be given in [u].

InOut:

Scope	Name	Type	Comment
Input	Enable	BOOL	Read continuously on "TRUE"
	Parameter-Number	WORD	Number of the parameter
Output	Valid	BOOL	Value is available
	Busy	BOOL	The function block with Busy = TRUE has control on the axis
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
	Value	BOOL	Value of the specified parameter in the data-type
Inout	Axis	Axis_Ref	Reference to axis

10.2.1.1.2.7 MC_ReadParameter (FB)

It will support the Parameters defined as “standard” in the PLCopen. These parameters will be transferred to the units as used for PLCopen function blocks



When a drive based axis implementation is used, the function block returns the value of a drive specific parameter or a PLCopen parameter. When the Central Motion Control axis implementation is used, a parameter according to the list PLCopen parameter is returned

No	Parameter Name	Data Type	Min.	Max.	Default	R/W	Comments
1	Commanded-Position	DINT				R	Commanded Position
2	SWLimitPos	DINT	-2147483647	2147483647	2147483647	R/W	Positive Software limit switch position
3	SWLimitNeg	DINT	-2147483647	2147483647	-2147483647	R/W	Negative Software limit switch position
4	EnableLimitPos	BOOL	FALSE	TRUE	FALSE	R/W	Enable positive software limit switch
5	EnableLimit-Neg	BOOL	FALSE	TRUE	FALSE	R/W	Enable negative software limit switch
6	EnablePostLag-Monitoring	BOOL	FALSE	TRUE	TRUE	R/W	Enable monitoring of position lag (following error)
7	MaxPosition-Lag	DINT	1	2147483647**		R	Maximal position lag
8	MaxVelocitySystem	DINT			32767	R	Maximal allowed velocity of the axis in the motion system
9	MaxVelocityAppl	DINT	0**	32767	32767	R/W	Maximal allowed velocity of the axis in the application
10	ActualVelocity	DINT	-32767	32767		R	Actual velocity
11	Commanded-Velocity	DINT	-32767	32767		R	Commanded Velocity
12	MaxAccelerationSystem	DINT			32767	R	Maximal allowed acceleration of the axis in the motion system

No	Parameter Name	Data Type	Min.	Max.	Default	R/W	Comments
13	MaxAccelerationAppl	DINT	10	32767	32767	R/W	Maximal allowed acceleration of the axis in the application
14	MaxDecelerationSystem	DINT			32767	R	Maximal allowed deceleration of the axis
15	MaxDecelerationAppl	DINT	10	32767	32767	R/W	Maximal allowed deceleration of the axis
16	MaxJerk	DINT	0*	2147483647	2147483647	R/W	Maximal allowed jerk of the axis
2001	MODULO_NOMINATOR	DINT	1	2147483647	1	R/W	ABB spec. parameter. Used f. Central Motion Control implementation: Gearbox modifier to MODULO_RANGE
2002	MODULO_DENOMINATOR	DINT	1	2147483647	1	R/W	ABB spec parameter. Used f. Central Motion Control implementation: Gearbox modifier to MODULO_RANGE
2003	EnableLimit2Decelerate	BOOL	FALSE	TRUE	FALSE	R/W	Enable software limit switches to decelerate
2004	EnableLimitAbort	BOOL	FALSE	TRUE	FALSE	R/W	Enable that software limit switches will abort ongoing movement
2005	EnableLimitVelocity	BOOL	FALSE	TRUE	FALSE	R/W	If the velocity is limited the unmoved position will be covered whenever possible
2006	SWLimit2DecPos	LREAL	-2147483647	2147483647	2147483647	R/W	Used as end position for EnableLimit2Decelerate
2007	SWLimit2DecNeg	LREAL	-2147483647	2147483647	2147483647	R/W	Used as end position for EnableLimit2Decelerate
2008	MaxPositionGapLL	LREAL	0	214748364700	0	R/W	Used to stop the ongoing movement if position is behind

*0 means: no limitation of jerk is performed.

**Axis will stay in stop.

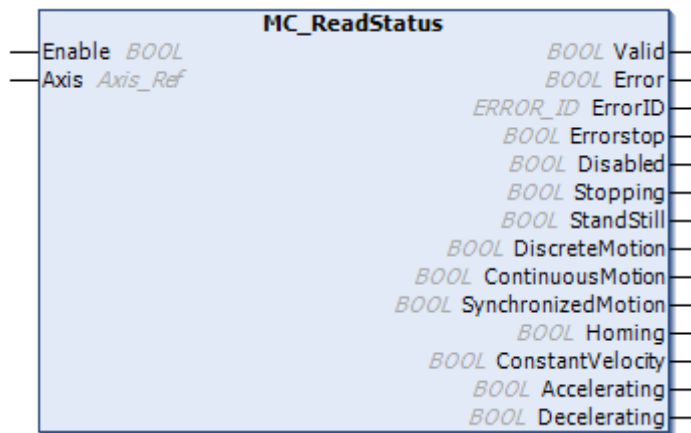
***Is modified by CMC_Axis_Control_Parameter, the maximum Value is calculated in increments, the value which is delivered by ReadParameter will be given in [u].

InOut:

Scope	Name	Type	Comment
Input	Enable	BOOL	Read continuously on “TRUE”
	Parameter-Number	WORD	Number of the parameter
Output	Valid	BOOL	Value is available
	Busy	BOOL	The function block with Busy = TRUE has control on the axis
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ER-ROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
	Value	DINT	Value of the specified parameter in the datatype
Inout	Axis	Axis_Ref	Reference to axis

10.2.1.1.2.8 MC_ReadStatus (FB)

This function block returns in detail the status of the axis with respect to the motion currently in progress.

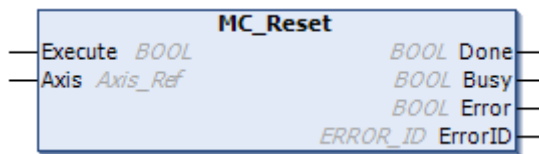


InOut:

Scope	Name	Type	Comment
Input	Enable	BOOL	Read continuously on "TRUE"
Output	Valid	BOOL	Value is available
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
	Errorstop	BOOL	Axis is in specific state
	Disabled	BOOL	Axis is in specific state
	Stopping	BOOL	Axis is in specific state
	StandStill	BOOL	Axis is in specific state
	DiscreteMotion	BOOL	Axis is in specific state
	ContinuousMotion	BOOL	Axis is in specific state
	SynchronizedMotion	BOOL	Axis is in specific state
	Homing	BOOL	Axis is in specific state
	ConstantVelocity	BOOL	Axis is in specific state, Motor moves with constant velocity
	Accelerating	BOOL	Axis is in specific state, Increasing energy of the motor
	Decelerating	BOOL	Axis is in specific state, Decreasing energy of the motor
Inout	Axis	Axis_Ref	Reference to axis

10.2.1.1.2.9 MC_Reset (FB)

MC_Reset function block makes the transition from the state ErrorStop to StandStill by resetting all internal axis-related errors. It does not affect the output of the function block instances



Gives a reset to the axis as well, in any state. In addition, a reset message is sent to the drive (e.g. output DRIVE_RESET_FAULT AT CMC_Basic_Kernel)



Output "Done" will be TRUE after a wait time of 5 seconds, even though Reset operation is complete.

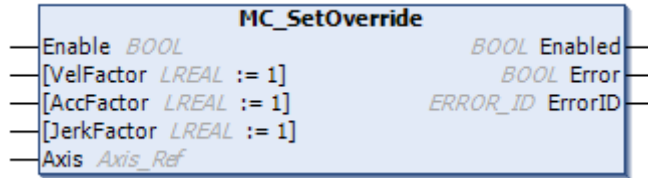
InOut:

Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
Output	Done	BOOL	Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BOOL	The function block with Busy = TRUE has control on the axis
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ER-ROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
Inout	Axis	Axis_Ref	Reference to axis

10.2.1.1.2.10 MC_SetOverride (FB)

This function block sets the values of override for the whole axis and all functions that are working on that axis.

The override parameters act as a factor that is multiplied to the commanded velocity, acceleration, deceleration and jerk of the move function block.



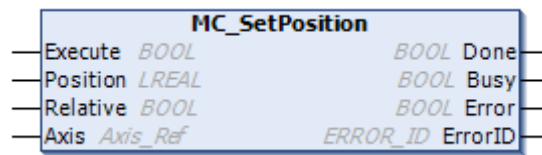
- The input AccFactor acts on positive and negative acceleration (deceleration).
- This function block sets the factor. The override factor is valid until a new override is set.
- The default values of the override factors are 1.0.
- The value of the overrides can be between 0.0 and 1.0. Values > 1.0 and values < 0.0 are not allowed. The value 0.0 is not allowed for AccFactor and JerkFactor.
- Override does not act on slave axes. (Axes in the state Synchronized Motion).
- The function block does not influence the single axis state diagram.
- The override factors are just effective to modify the velocity, acceleration, deceleration and jerk which are provided as explicit values by PLCopen function blocks. They do not modify a movement which is commanded by other means as camming, gearing, profiling, where no explicit velocity, acceleration, deceleration and jerk is in place

InOut:

Scope	Name	Type	Initial	Comment
Input	Enable	BOOL		If TRUE, it writes the value of the override factor continuously. If FALSE it should keep the last value
	VelFactor	LREAL	1	New override factor for the velocity, Range: =0.0, >0.0, <1.0
	AccFactor	LREAL	1	New override factor for the acceleration or deceleration, Range: >0.0, < 1.0
	JerkFactor	LREAL	1	New override factor for the jerk, Range: >0.0, <1.0
Output	Enabled	BOOL		The block is enabled and working
	Error	BOOL		Signals that error has occurred within function block
	ErrorID	ERROR_ID		Error identification. For error details refer to Enumeration ERROR_ID
Inout	Axis	Axis_Ref		Reference to axis

10.2.1.1.2.11 MC_SetPosition (FB)

This function block shifts the coordinate system of an axis by manipulating both the set-point position as well as the actual position of an axis with the same value without any movement caused.



(Re-calibration with same following error). This can be used for instance for a reference situation. This function block can also be used during motion without changing the commanded position, which is now positioned in the shifted coordinate system. This block may just be called in: “StandStill”, “ContinuousMotion”, “ErrorStop” or “Disabled”.

- **RELATIVE** means that position is added to the actual position value of the axis at the time of execution. This results in a recalibration by a specified distance.
- **ABSOLUTE** means that the actual position value of the axis is set to the value specified in the position parameter.

InOut:

Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge.
	Position	LREAL	[u] Reference position, used as Distance with Relative = TRUE
	Relative	BOOL	TRUE = Relative, FALSE = Absolute
Output	Done	BOOL	Execution is finished
	Busy	BOOL	The function block is not finished
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
Inout	Axis	Axis_Ref	Reference to axis

10.2.1.1.2.12 MC_WriteBoolParameter (FB)

It will support the Parameters defined as “standard” in the PLCOpen. These parameters will be transferred to the units as used for PLCOpen function blocks



When a drive based axis implementation is used, the Function Block writes the value of a drive specific BOOL parameter or a PLCOpen BOOL parameter. When the Central Motion Control axis implementation is used, a BOOL parameter according to the list PLCOpen parameter is written

No	Parameter Name	Data Type	Min.	Max.	Default	R/W	Comments
1	CommandedPosition	DINT				R	Commanded Position
2	SWLimitPos	DINT	-2147483647	2147483647	2147483647	R/W	Positive Software limit switch position
3	SWLimitNeg	DINT	-2147483647	2147483647	-2147483647	R/W	Negative Software limit switch position
4	EnableLimitPos	BOOL	FALSE	TRUE	FALSE	R/W	Enable positive software limit switch
5	EnableLimitNeg	BOOL	FALSE	TRUE	FALSE	R/W	Enable negative software limit switch
6	EnablePostLagMonitoring	BOOL	FALSE	TRUE	TRUE	R/W	Enable monitoring of position lag (following error)
7	MaxPositionLag	DINT	1	2147483647** *		R	Maximal position lag
8	MaxVelocitySystem	DINT			32767	R	Maximal allowed velocity of the axis in the motion system
9	MaxVelocityAppl	DINT	0**	32767	32767	R/W	Maximal allowed velocity of the axis in the application
10	ActualVelocity	DINT	-32767	32767		R	Actual velocity

No	Parameter Name	Data Type	Min.	Max.	Default	R/W	Comments
11	CommandedVelocity	DINT	-32767	32767		R	Commanded Velocity
12	MaxAccelerationSystem	DINT			32767	R	Maximal allowed acceleration of the axis in the motion system
13	MaxAccelerationAppl	DINT	10	32767	32767	R/W	Maximal allowed acceleration of the axis in the application
14	MaxDecelerationSystem	DINT			32767	R	Maximal allowed deceleration of the axis
15	MaxDecelerationAppl	DINT	10	32767	32767	R/W	Maximal allowed deceleration of the axis
16	MaxJerk	DINT	0*	2147483647	2147483647	R/W	Maximal allowed jerk of the axis
2001	MODULO_NOMINATOR	DINT	1	2147483647	1	R/W	ABB specific parameter. Used for Central Motion Control implementation: Gearbox modifier to MODULO_RANGE
2002	MODULO_DENOMINATOR	DINT	1	2147483647	1	R/W	ABB specific parameter. Used for Central Motion Control implementation: Gearbox modifier to MODULO_RANGE
2003	EnableLimit2Decelerate	BOOL	FALSE	TRUE	FALSE	R/W	Enable software limit switches to decelerate
2004	EnableLimitAbort	BOOL	FALSE	TRUE	FALSE	R/W	Enable that software limit switches will abort ongoing movement
2005	EnableLimitVelocity	BOOL	FALSE	TRUE	FALSE	R/W	If the velocity is limited the unmoved position

No	Parameter Name	Data Type	Min.	Max.	Default	R/W	Comments
							tion will be covered whenever possible
2006	SWLimit2DecPos	LREAL	-2147483647	2147483647	2147483647	R/W	Used as end position for EnableLimit2Decelerate
2007	SWLimit2DecNeg	LREAL	-2147483647	2147483647	2147483647	R/W	Used as end position for EnableLimit2Decelerate
2008	MaxPosition-GapLL	LREAL	0	21474836470	0	R/W	Used to stop the ongoing movement if position is behind

*0 means: no limitation of jerk is performed.

**Axis will stay in stop.

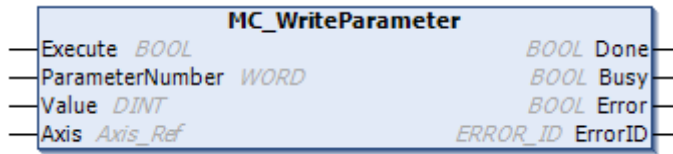
***Is modified by CMC_Axis_Control_Parameter, the maximum Value is calculated in increments, the value which is delivered by ReadParameter will be given in [u].

InOut:

Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
	ParameterNumber	WORD	Number of the parameter (correspondence between number and parameter is to be specified later)
	Value	BOOL	New value of the specified parameter
Output	Done	BOOL	Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BOOL	The function block is not finished
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
Inout	Axis	Axis_Ref	Reference to axis

10.2.1.1.2.13 MC_WriteParameter (FB)

It will support the Parameters defined as “standard” in the PLCopen. These parameters will be transferred to the units as used for PLCopen function blocks



When a drive based axis implementation is used, the function block writes the value of a drive specific parameter or a PLCopen parameter. When the Central Motion Control axis implementation is used, a parameter according to the list PLCopen parameter is written

No	Parameter Name	Data-Type	Min.	Max.	Default	R/W	Comments
1	Commanded-Position	DINT				R	Commanded Position
2	SWLimitPos	DINT	-2147483647	2147483647	2147483647	R/W	Positive Software limit switch position
3	SWLimitNeg	DINT	-2147483647	2147483647	-2147483647	R/W	Negative Software limit switch position
4	EnableLimit-Pos	BOOL	FALSE	TRUE	FALSE	R/W	Enable positive software limit switch
5	EnableLimit-Neg	BOOL	FALSE	TRUE	FALSE	R/W	Enable negative software limit switch
6	EnablePost-LagMonitoring	BOOL	FALSE	TRUE	TRUE	R/W	Enable monitoring of position lag (following error)
7	MaxPosition-Lag	DINT	1	2147483647**		R	Maximal position lag
8	MaxVelocitySystem	DINT			32767	R	Maximal allowed velocity of the axis in the motion system
9	MaxVelocityAppl	DINT	0**	32767	32767	R/W	Maximal allowed velocity of the axis in the application
10	ActualVelocity	DINT	-32767	32767		R	Actual velocity
11	Commanded-Velocity	DINT	-32767	32767		R	Commanded Velocity
12	MaxAccelerationSystem	DINT			32767	R	Maximal allowed acceleration of the axis in the motion system
13	MaxAccelerationAppl	DINT	10	32767	32767	R/W	Maximal allowed acceleration of the axis in the application
14	MaxDecelerationSystem	DINT			32767	R	Maximal allowed deceleration of the axis
15	MaxDecelerationAppl	DINT	10	32767	32767	R/W	Maximal allowed deceleration of the axis

No	Parameter Name	Data-Type	Min.	Max.	Default	R/W	Comments
16	MaxJerk	DINT	0*	2147483647	2147483647	R/W	Maximal allowed jerk of the axis
2001	MODULO_NO-MINATOR	DINT	1	2147483647	1	R/W	ABB spec.parameter. Used f.Central Motion Control implementation: Gearbox modifier to MODULO_RANGE
2002	MODULO_DE-NOMINATOR	DINT	1	2147483647	1	R/W	ABB spec. parameter. Used f. Central Motion Control implementation: Gearbox modifier to MODULO_RANGE
2003	EnableLimit2Decelerate	BOOL	FALSE	TRUE	FALSE	R/W	Enable software limit switches to decelerate
2004	EnableLimitAbort	BOOL	FALSE	TRUE	FALSE	R/W	Enable that software limit switches will abort ongoing movement
2005	EnableLimitVelocity	BOOL	FALSE	TRUE	FALSE	R/W	If the velocity is limited the unmoved position will be covered whenever possible
2006	SWLimit2DecPos	LREAL	-2147483647	2147483647	2147483647	R/W	Used as end position for EnableLimit2Decelerate
2007	SWLimit2DecNeg	LREAL	-2147483647	2147483647	2147483647	R/W	Used as end position for EnableLimit2Decelerate
2008	MaxPosition-GapLL	LREAL	0	214748364700	0	R/W	Used to stop the ongoing movement if position is behind

*0 means: no limitation of jerk is performed.

**Axis will stay in stop.

***Is modified by CMC_Axis_Control_Parameter, the maximum Value is calculated in increments, the value which is delivered by ReadParameter will be given in [u].

InOut:

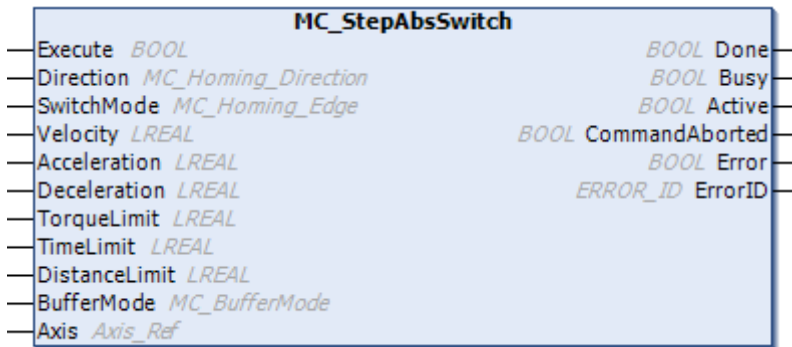
Scope	Name	Type	Comment
Input	Execute	BO OL	Starts the function block at rising edge
	Parameter- Number	W OR D	Number of the parameter (correspondence between number and parameter is to be specified later)
	Value	DI NT	New value of the specified parameter
Output	Done	BO OL	Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BO OL	The function block is not finished
	Error	BO OL	Signals that error has occurred within function block
	ErrorID	ER RO R_I D	Error identification. For error details refer to Enumeration ERROR_ID
Input	Axis	Axi s_ Re f	Reference to axis

10.2.1.1.3 MC Homing

10.2.1.1.3.1 MC_StepAbsSwitch (FB)

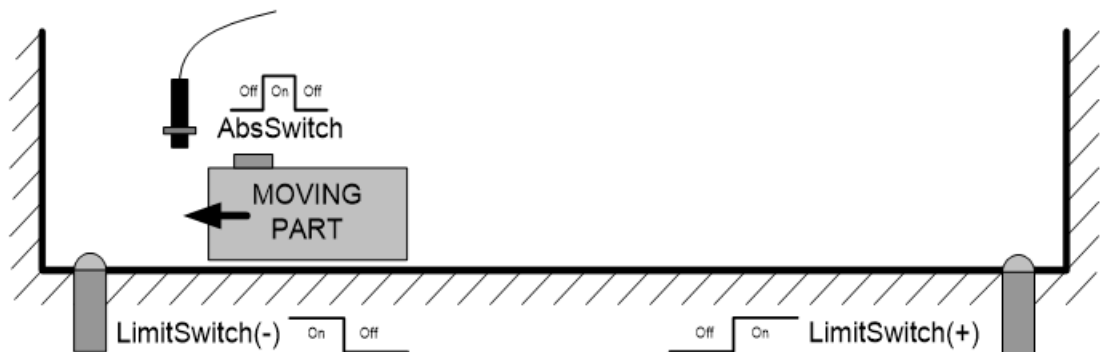
This function block performs a homing function by searching for an absolute positioned external physical switch. An Absolute Switch has two “Off” or “On” areas, see example.

For central Motion Control implementation: The signal of the Absolute Switch has to be written to the variable “absRefSwitch” of the data type CMC_Axis_IO



Inside the operation area the limit switches have to be logically FALSE and outside the borders the signal of the corresponding limit switch has to be logically TRUE. If needed the signal from the sensor must be inverted before it is connected to an element the CMC_Axis_IO data type

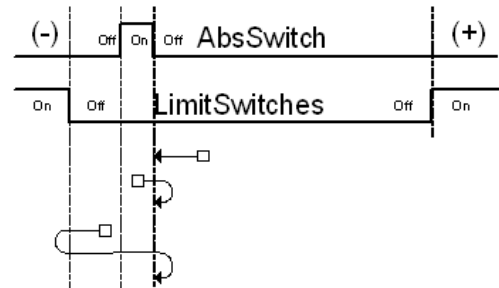
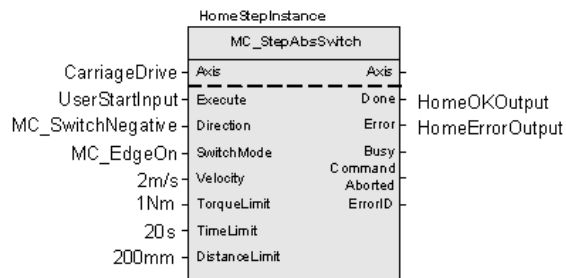
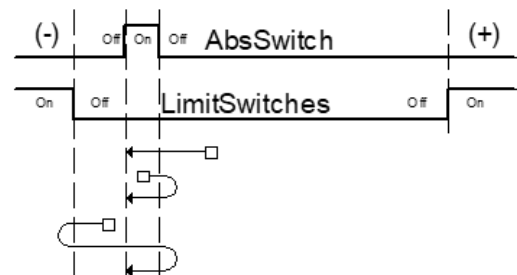
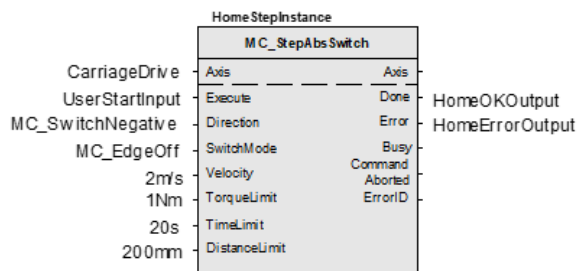
Example



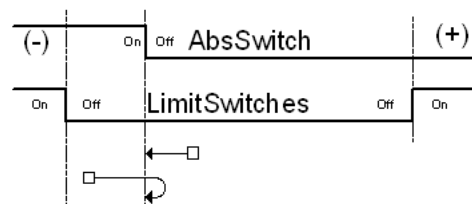
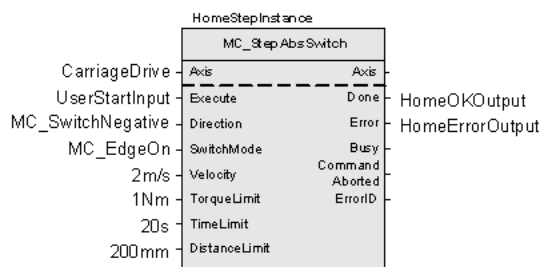
This physical layout has the risk that homing is started in the wrong direction (escaping the switch). To support such case, it implements a special behavior when Limit Switches are found (or the AbsSwitch itself is “On” at Execute)-

- Axis State is set to Homing,
- The homing is commanded in the most likely direction were the sensor can be found. In this example(-),
- The velocity is defined by the input,
- Both time and distance limits can cause an error if exceeded,
- If any LimitSwitch is found during Homing (any of them), then a special process is started in the opposite direction, the AbsSwitch is searched to switch off (or “On” depending on SwitchMode setting). The Edge (passed by), and homing process is restarted in the original direction and with the same conditions. This ensures that the end conditions are always same,

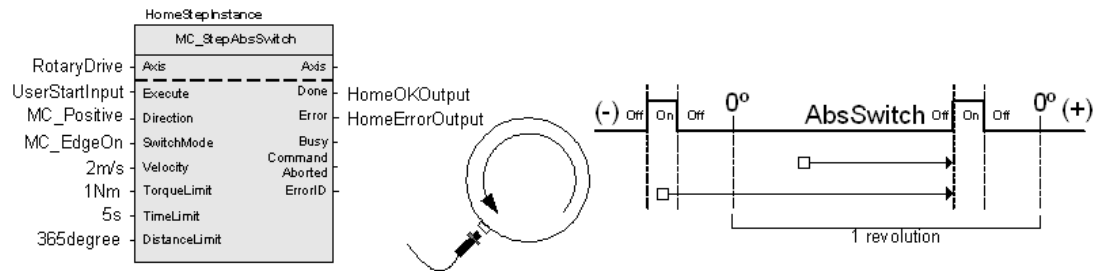
- If the SwitchMode is either MC_SwitchNegative or MC_SwitchPositive, then the special process is also started in opposite direction depending from the switch state at “Execute”,
- The direction changes only when the specified Velocity is reached (InVelocity),
- This function block does not modify the actual position,
- This function block does not leave the Homing State when done.
- This function block can only be used once for a homing sequence



An overlapping switch configuration is also possible. This has same the behavior as working on the limit switches



If the input direction is set to a fixed direction (MC_Positive or MC_Negative), then the initial switch state is ignored (used for example in rotary axis where only one sense of rotation is allowed):



With an overlapping switch configuration either MC_EdgeOn or MC_EdgeOff can be used for the input SwitchMode. This depends on the switching behaviour of the absolute switch and the used option for the input Direction

InOut:

Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
	Direction	MC_Homing_Direction	Specifies the direction of the motion if any: MC_Positive = Starts in positive direction always MC_Negative = Starts in negative direction always MC_SwitchPositive = Depends on Switch status at Execute edge. If Switch is "Off", direction is positive, if "On" it is negative MC_SwitchNegative = Like previous, but opposite
	SwitchMode	MC_Homing_Edge	Sensor condition to finalize this function block in any switch mode: MC_On = When sensor is ON. MC_Off = When sensor is OFF. MC_EdgeOn = When Off to On transition in sensor MC_EdgeOff = When On to Off transition in sensor
	Velocity	LREAL	[u/s] Value of the maximum velocity (not necessarily reached). Range: >0
	Acceleration	LREAL	[u/s ²] Value of the acceleration (increasing energy of the motor). Range: >0
	Deceleration	LREAL	[u/s ²] Value of the deceleration (decreasing energy of the motor). Range: >0
	TorqueLimit	LREAL	Not supported
	TimeLimit	LREAL	[s] If the function block condition is not met in the TimeLimit, an error is issued. <=0: No time limit
	DistanceLimit	LREAL	[u] If the function block condition is not met within a DistanceLimit travel, an error is issued. 0 = No distance limit

Scope	Name	Type	Comment
	BufferMode	MC_BufferMode	Not supported, default mcABORTING used
Output	Done	BOOL	Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BOOL	The function block is not finished
	Active	BOOL	Indicates that the function block has control on the axis
	CommandAborted	BOOL	Command is aborted by another command from other PLCopen function block
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID Specific Error numbers: + MC_TimeLimitExceeded + MC_DistanceLimitExceeded + MC_TorqueLimitExceeded
Inout	Axis	Axis_Ref	Reference to axis

10.2.1.1.3.2 MC_StepDirect (FB)

This function Block performs a static homing by directly forcing an actual position. No physical motion is performed in this mode. This is equivalent to a MC_SetPosition action, but clears the Homing State.



This function block modifies actual position and sets to the “SetPosition” input value at the end.

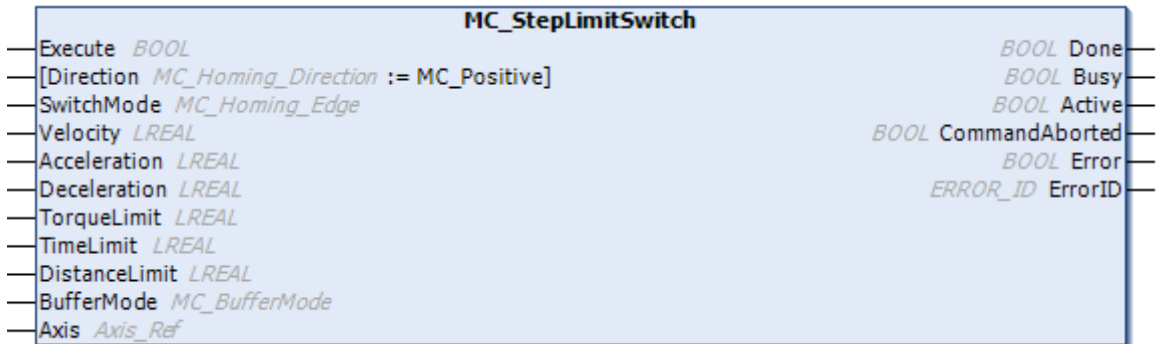
InOut:

Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
	SetPosition	LREAL	[u] Value of the absolute position to be set when homing is done
	BufferMode	MC_Buffer-Mode	not supported, default mcABORTING used
Output	Done	BOOL	Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BOOL	The function block is not finished
	Active	BOOL	Indicates that the function block has control on the axis
	CommandAborted	BOOL	Command is aborted by another command from other PLCopen function block
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
Inout	Axis	Axis_Ref	Reference to axis

10.2.1.1.3.3 MC_StepLimitSwitch (FB)

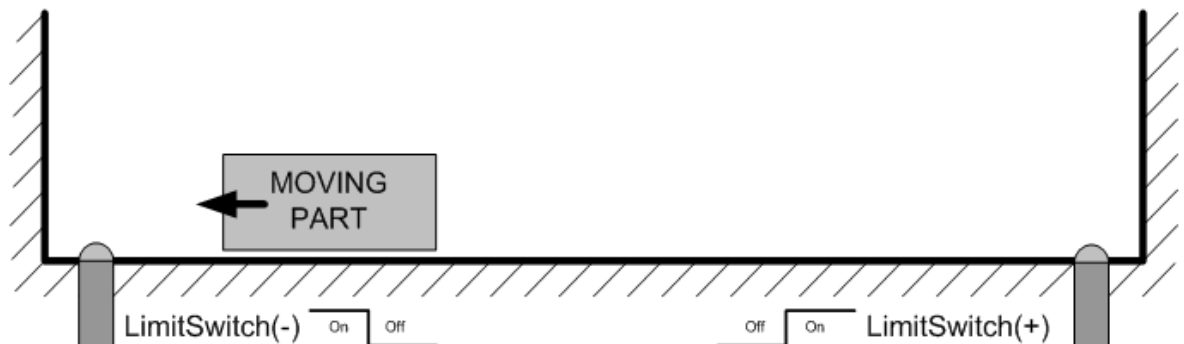
This function block performs a homing function by searching for sensor using only limit switches. A limit switch has one “Off” or “On” area.

The signal of the Limit Switches have to be written to the variables “limitSwitchPos” and “limitSwitchNeg” of the data type CMC_Axis_IO



Inside the operation area the limit switches have to be logically FALSE and outside the borders the signal of the corresponding limit switch has to be logically TRUE. If needed the signal from the sensor must be inverted before it is connected to an element the AXIS_IO data type

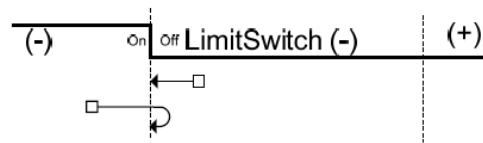
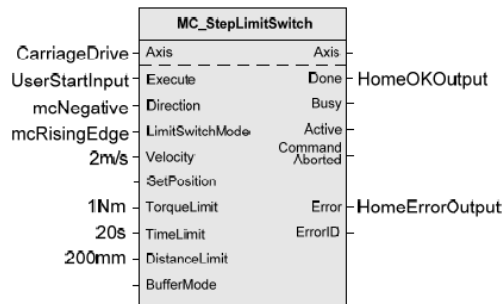
Example



In this case the limit switches (always active once moving part working area has been surpassed) are used for homing procedure.

- The axis State is changed to Homing.
- Home is commanded by user in the desired homing direction at the selected Velocity.
- If LimitSwitch is found “On” on rising “Execute”, then the process is started in the opposite direction as specified, LimitSwitch is search for “Off” (or On depending in LimitSwitchMode setting) Edge (released), and process is restarted again in original direction. This ensures that the end conditions are always the same.
- The time and distance limits can cause error if exceeded.
- The direction changes only when the specified velocity is reached, this ensures acceleration and deceleration spaces are fixed.
- This function block does not modify actual position.
- This function block does not leave the Homing State when done.

- This function block can only be used once for a homing sequence



In-

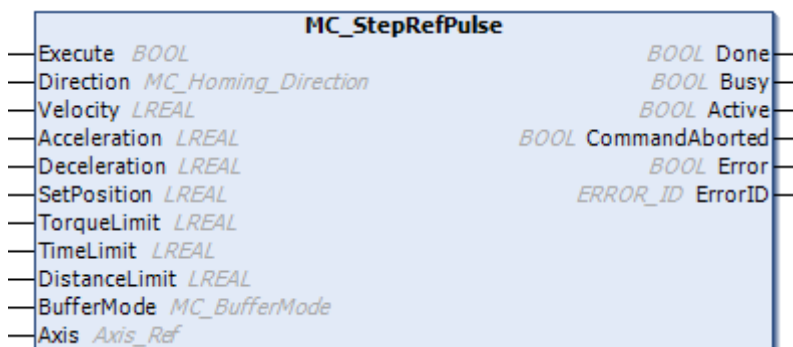
Out:

Scope	Name	Type	Initial	Comment
Input	Execute	BOOL		Starts the function block at rising edge
	Direction	MC_Homing_Direction	MC_Positive	Specifies the direction of the motion and corresponding LimitSwitch to search for, just MC_Positive and MC_Negative are possible: MC_Positive = Positive direction searching positive LimitSwitch MC_Negative = Negative direction searching negative LimitSwitch
	Switch-Mode	MC_Homing_Edge		Sensor condition to finalize this function block: MC_On = When sensor is ON. MC_EdgeOn = When Off to On transition in sensor
	Velocity	LREAL		[u/s] Value of the maximum velocity (not necessarily reached). Range: >0
	Acceleration	LREAL		[u/s ²] Value of the acceleration (increasing energy of the motor). Range: >0. If value = 0, Acceleration will be equal to parameter paraMaxAccelerationAppl
	Deceleration	LREAL		[u/s ²] Value of the deceleration (decreasing energy of the motor). Range: >0. If value = 0, Deceleration will be equal to parameter paraMaxAccelerationAppl
	TorqueLimit	LREAL		[u] Not supported, switched off by default (Maximum torque or force. <0 = No torque limit)
	TimeLimit	LREAL		[s] If the function block condition is not met in the TimeLimit, an error is issued. <=0: No time limit.
	DistanceLimit	LREAL		[u] If the function block condition is not met within a DistanceLimit travel, an error is issued. <0: No distance limit.

Scope	Name	Type	Initial	Comment
	Buffer-Mode	MC_BufferMode		Not supported, default mcABORTING used
Output	Done	BOOL		Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BOOL		The function block is not finished
	Active	BOOL		Indicates that the function block has control on the axis
	CommandAborted	BOOL		Command is aborted by another command from other PLCopen function block
	Error	BOOL		Signals that error has occurred within function block
	ErrorID	ERROR_ID		Error identification. For error details refer to Enumeration ERROR_ID Specific Error numbers: + MC_TimeLimitExceeded + MC_DistanceLimitExceeded + MC_TorqueLimitExceeded
Inout	Axis	Axis_Ref		Reference to axis

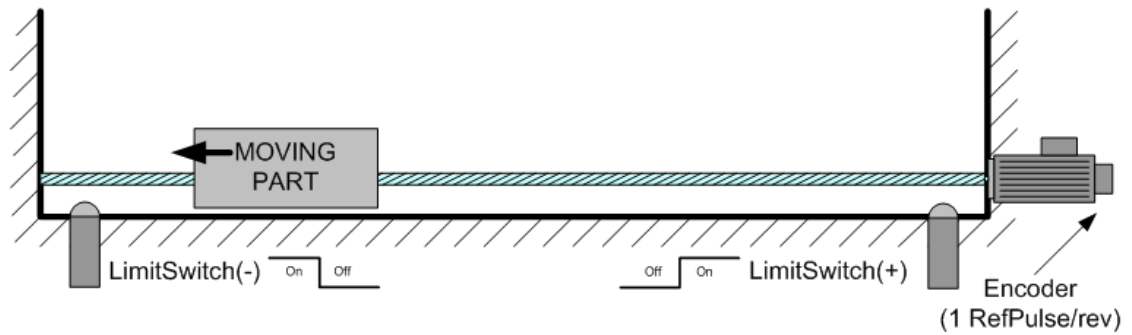
10.2.1.1.3.4 MC_StepRefPulse (FB)

This function block performs homing by searching for Zero pulse (also called Marker or reference pulse) in encoder.



The reference pulse appears once per encoder revolution. The advantage in using Reference Pulse for homing is the higher accuracy and precision that can be achieved compared to traditional optical, mechanical or magnetic sensors.

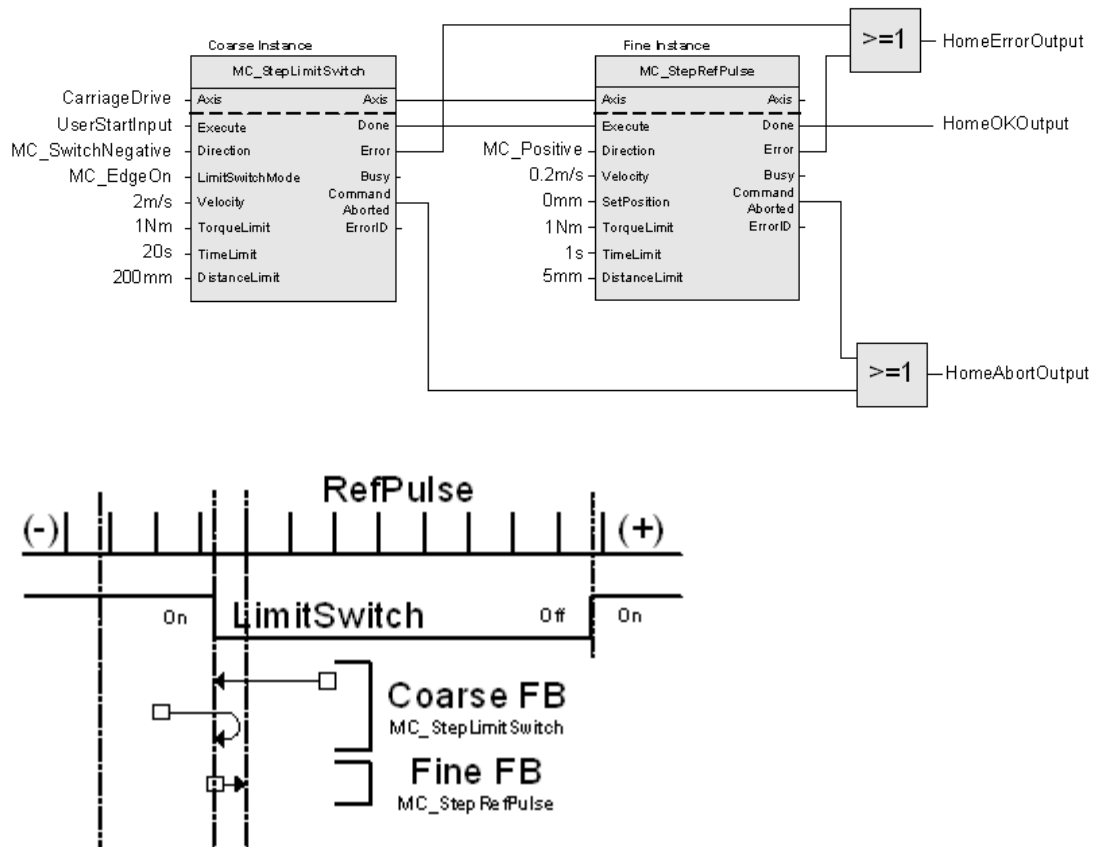
Example



The axis state is changed to Homing if not already in.

- Home is commanded by user in the desired homing direction at the programmed velocity.
- First occurrence of the Reference Pulse, Homing is finished.
- Torque is limited. Time and Distance Limits can cause error if exceeded.
- This function block modifies actual position and sets to the "SetPosition" input value at the end
- This function block clears the Homing State when Done.

It is common that a first approach is performed against a mechanical sensor at higher velocity, and after a Reference Pulse, at a lower velocity. This is a traditional 2-Step homing (Coarse by external Switch in reverse and Fine by Reference Pulse in forward). For ease of use both functions could be grouped together in single function block. Advantage having the function blocks separate is that any combination is possible (MC_Block and after MC_RefPulse, etc.), stating different velocity and conditions for each Step (highly flexible), without increasing homing function block complexity too much.



InOut:

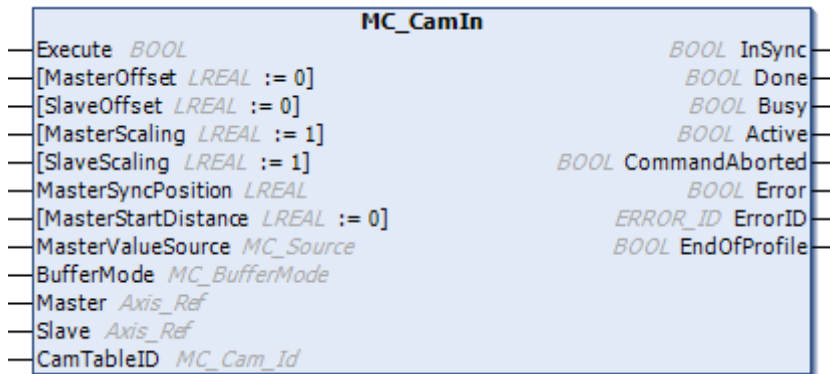
Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
	Direction	MC_Ho- ming_Direc- tion	Specifies the direction to start the motion, just MC_Positive and MC_Negative are possible to use: MC_Positive = Starts in positive direction always MC_Negative = Starts in negative direction always
	Velocity	LREAL	[u/s] Value of the maximum velocity (not necessarily reached). Range: >0
	Acceleration	LREAL	[u/s ²] Value of the acceleration (increasing energy of the motor). Range: >0

Scope	Name	Type	Comment
	Deceleration	LREAL	[u/s ²] Value of the deceleration (decreasing energy of the motor). Range: >0
	SetPosition	LREAL	[u] Value of the absolute position to be set when homing is done
	TorqueLimit	LREAL	[u] Maximum torque or force. 0 = No torque limit
	TimeLimit	LREAL	[s] If the function block condition is not met in the TimeLimit, an error is issued. 0 = No time limit
	DistanceLimit	LREAL	[u] If the function block condition is not met within a DistanceLimit travel, an error is issued. 0 = No distance limit
	BufferMode	MC_Buffer-Mode	Not supported, default mcABORTING used
Output	Done	BOOL	Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BOOL	The function block is not finished
	Active	BOOL	Indicates that the function block has control on the axis
	CommandAborted	BOOL	Command is aborted by another command from other PLCopen function block
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID Specific Error numbers: + MC_TimeLimitExceeded + MC_DistanceLimitExceeded + MC_TorqueLimitExceeded
Inout	Axis	Axis_Ref	Reference to axis

10.2.1.1.4 MC MultiAxis

10.2.1.1.4.1 MC_CamIn (FB)

This function block implements Camming functionality. A slave axis is coupled to a master axis by a position/position relation.



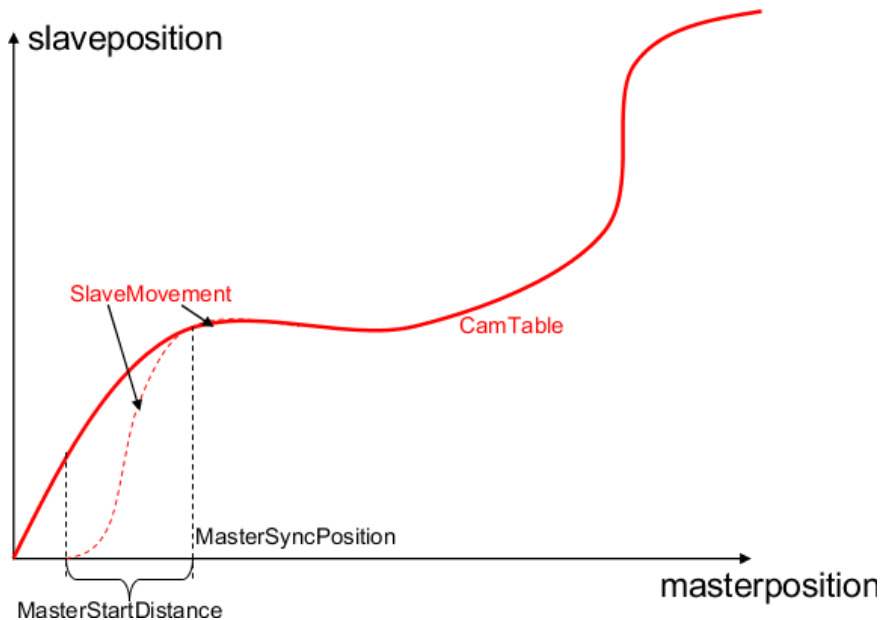
- It is not required that the master is stationary.
- If the actual master and slave positions do not correspond to the offset values when MC_CamIn is executed, either an error occurs or the system deals with the difference automatically.
- The Cam is placed either absolute or relative to the current master and slave positions:
- Absolute: The profile between master and slave is seen as an absolute relationship.
- Relative: The relationship between master and slave is in a relative mode.
- If a cam-table is to be used “relative”, the first position has to be zero.
- This function block is not merged with the MC_CamTableSelect function block because this separation enables changes on the fly.
- A mechanical analogy to a slave offset is a cam welded with additional constant layer thickness. Because of this the slave positions have a constant offset and the offset could be interpreted as axis offset of the master shaft, if linear guided slave tappets are assumed.



The slave axis is NOT ramped out, which means the curve should end with velocity=0. The CAM could be interrupted with any other function block, according to the statemachine. It is not required to use MC_CamOut.

In case MasterSyncPosition and MasterStartDistance are equal to 0 (default value), the Cam-Function is started with the rising edge of "Execute". When the master did not yet reach a position in the cam area, the slave axis is stopped.

With MasterStartDistance \neq 0 the slave axis needs to be in state StandStill when activating MC_CamIn. The function block will wait until the master axis reaches the position MasterSyncPosition - MasterStartDistance. The slave will then be started and be synchronized to the CAM table, to the position and velocity which is indicated by the master position MasterSyncPosition.



	The MC_CamIn has parameters to scale the cam-table values (MasterScaling, SlaveScaling). It has to be considered that MasterOffset and SlaveOffset are scaled exactly like the corresponding cam-table values. The MasterSyncPosition and MasterStartDistance are not scaled at all, these positions are related to the actual master position whereas the MasterOffset and SlaveOffset are related to the camtable. New set of values at inputs MasterOffset, SlaveOffset, MasterScaling, SlaveScaling will be accepted only after the function block is aborted and fresh rising edge is provided at Execute input.
	The default behavior of this function block can be modified by the inputs in function MCA_Cam_Extra
	A negative MasterScaling requires backward master movement, when combined with MasterOffset. The MasterScaling also applies to the MasterOffset. Behaviour results from the requirement to have ascending master values in CamTable.
	User can utilize CAM editor in Automation builder to generate Cam table (MC_PProfile) automatically. For more details refer to Automation builder help.

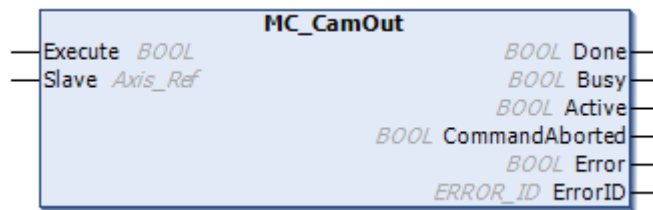
InOut:

Scope	Name	Type	Initial	Comment
Input	Execute	BOOL		Starts the function block at rising edge
	MasterOffset	LREAL	0	Offset of master table. Angular offset of the master shaft to Cam. Actual position - MasterOffset will be used to sample the CamTable
	SlaveOffset	LREAL	0	Offset of slave table. Sharpened cam (i.e higher elevation and deeper depression). Use the result from CamTable + SlaveOffset as reference position
	MasterScaling	LREAL	1	Scaling factor for master positions in CamTable. From the slave point of view the master overall profile is multiplied by this factor
	SlaveScaling	LREAL	1	Scaling factor for slave positions from CamTable. The overall slave profile is multiplied by this factor
	MasterSyncPosition	LREAL		Slave axis should be synchronized to CamTable when master axis reaches MasterSyncPosition
	MasterStartDistance	LREAL	0	If 0 : Start with rising edge of execute, >0 : Wait for MasterSyncPosition - MasterStartDistance to start the CamTable
	MasterValueSource	MC_Source		Defines the source for synchronization: mcSetValue - Synchronization on master set value. mcActualValue - Synchronization on master actual value
	BufferMode	MC_Buffer-Mode		Not supported, default mcABORTING used
Output	InSync	BOOL		Slave is synchronized to CamTable
	Done	BOOL		Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BOOL		The function block with Busy = TRUE has control on axis
	Active	BOOL		Indicates that the function block has control on the axis
	CommandAborted	BOOL		Command is aborted by another command from other PLCopen function block
	Error	BOOL		Signals that error has occurred within function block
	ErrorID	ERROR_ID		Error identification. For error details refer to Enumeration ERROR_ID
	EndOfProfile	BOOL		Pulsed output signaling the cyclic end of the CAM profile. It is displayed every time when the end of CAM profile is reached. In reverse direction, the 'EndOfProfile' is displayed also at the end of the cam profile (in this case the first point of the cam profile)
Inout	Master	Axis_Ref		Reference to master axis

Scope	Name	Type	Initial	Comment
	Slave	Axis_Ref		Reference to slave axis
	CamTableID	MC_Cam_Id		Identifier of CAM Table to be used in the MC_CamIn function block. Prepared by MC_CamTableSelect



10.2.1.1.4.2 MC_CamOut (FB)

This function block disengages the Slave axis from the Master axis immediately



The slave axis will keep the actual velocity.

It is assumed that this command is followed by another command, for instance MC_Stop, MC_GearIn, or any other command. If there is no new command, the default condition should be: Maintain Last Velocity. If there is no new command the axis will maintain its last velocity.

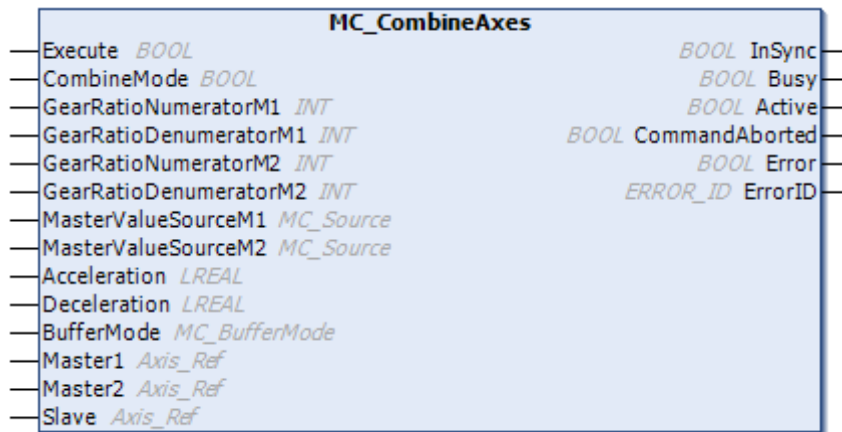
	It is not required to use this block to disengage the cam. MC_Stop/MC_Halt or any other command could be used instead.
	User can utilize CAM editor in Automation builder to generate Cam table (MC_PProfile) automatically. For more details refer to Automation builder help.

InOut:

Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
Output	Done	BOOL	Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BOOL	The function block with Busy = TRUE has control on the axis
	Active	BOOL	Indicates that the function block has control on the axis
	CommandAborted	BOOL	Command is aborted by another command from other PLCopen function block
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
Inout	Slave	Axis_Ref	Reference to slave axis

10.2.1.1.4.3 MC_CombineAxes (FB)

This function block combines the motion of 2 axes into a third axis with selectable combination method.

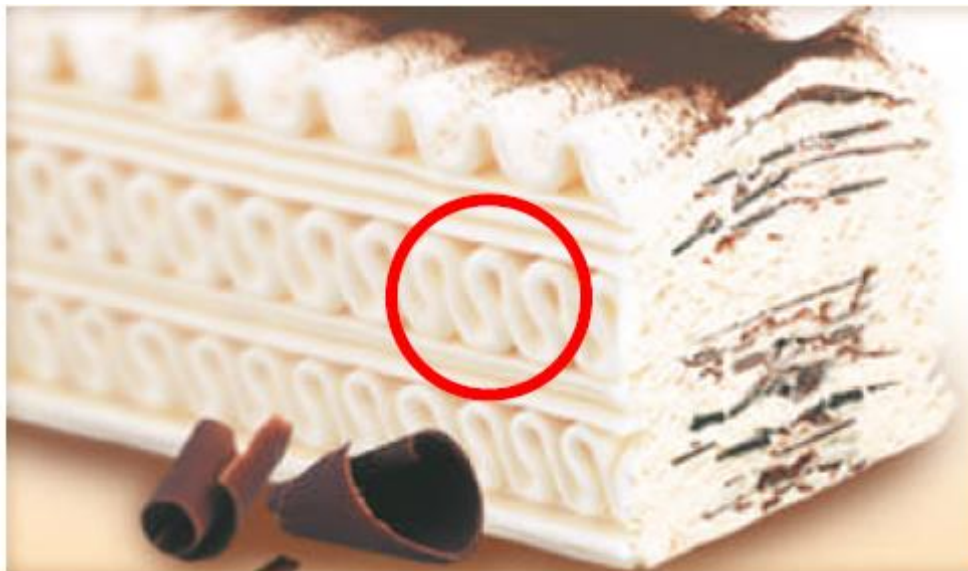


Basically it is a calculation of a new position setpoint based on the two position setpoints of the input axes. This function block is reflected in the state diagram like a synchronized motion type. As application example one can work with a separate profile synchronized to an object on a moving belt, or a rotating knife with flexible covered distance to be cut.



To stop the motion, the function block has to be interrupted by another function block issuing a new command.

Example of Ice-cream

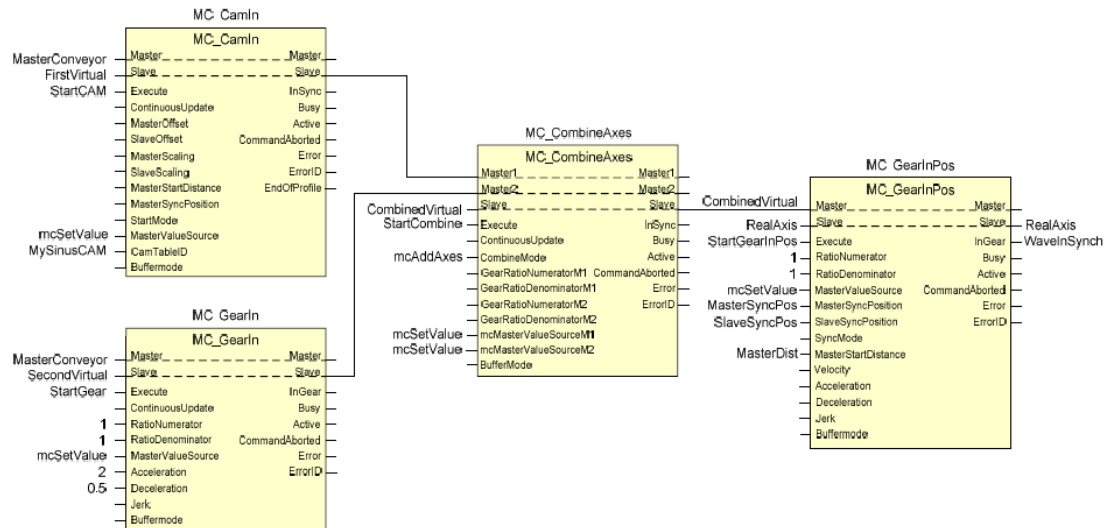


MC_CombineAxes can generate special synchronized movements that are not possible or complex to generate in other ways. In the following example, a CAM function block and the result of a Gear function block are both synchronized to a conveyor master, are added to generate a virtual master for a MC_GearInPos function of the final axis that will execute the movement. The particular application of this example could be a machine to deposit the icecream waving layers on top of the icecream base travelling through the freezer line in icecream factory. The dosing axis has to synchronize with a waving manner to the conveyor carrying the icecream base block. And it has to do this in a particular starting position and wave phase to achieve the expected result (therefore the

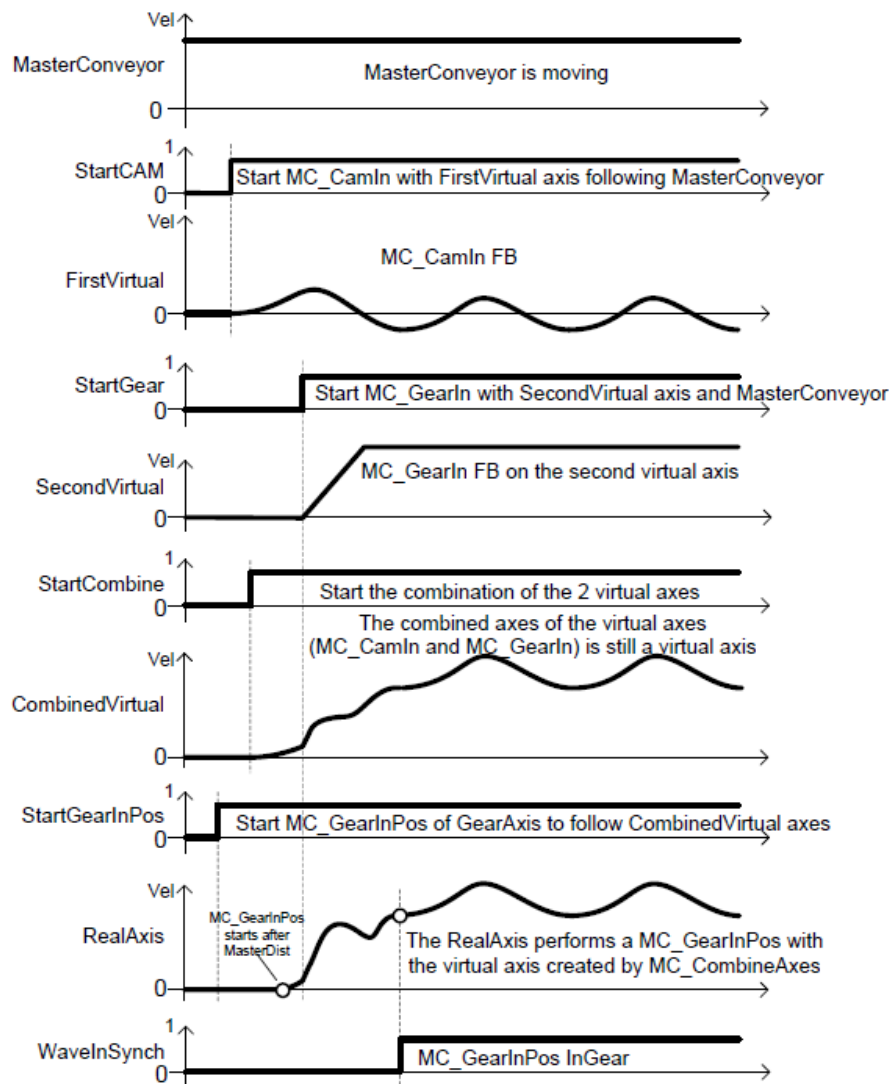
GearInPos). With the CAM function block one can define different wave patterns easily (like the one longer in the top of icecream).

Another case application can be chocolate bars with decoration (individual bars in mouldings). The dosificator makes the wave synchronized with conveyor and returns for the next.

Application example of MC_CombineAxes



The corresponding timing diagram for MC_CombineAxes example



This block has to be called within the REAL-TIME task, same task as CMC_BASIC_KERNEL

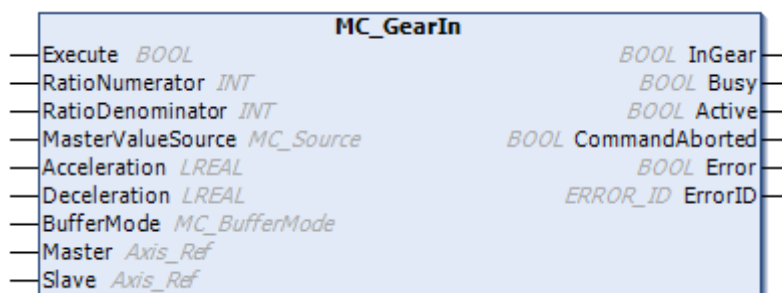
InOut:

Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
	Combine-Mode	BOOL	FALSE = Addition of two input axes positions, TRUE = Subtraction of two input axes positions
	GearRatio-NumeratorM1	INT	Gear ratio numerator for master axis 1 towards the slave
	GearRatio-DenominatorM1	INT	Gear ratio denominator for master axis 1 towards the slave
	GearRatio-NumeratorM2	INT	Gear ratio numerator for master axis 2 towards the slave
	GearRatio-DenominatorM2	INT	Gear ratio denominator for master axis 2 towards the slave
	MasterValueSourceM1	MC_Source	Decide to use the actual position or reference position of master axis 1. mcSetValue: Synchronization on master set value. mcActualValue: Synchronization on master actual value
	MasterValueSourceM2	MC_Source	Decide to use the actual position or reference position of master axis 2. mcSetValue: Synchronization on master set value. mcActualValue: Synchronization on master actual value
	Acceleration	LREAL	[u/s ²] Value of the acceleration (increasing energy of the motor), just applied until "insync" is reached. Range: >0
	Deceleration	LREAL	[u/s ²] Value of the deceleration (decreasing energy of the motor), just applied until "insync" is reached. Range: >0
	BufferMode	MC_Buffer-Mode	Not supported, default mcABORTING used
	InSync	BOOL	Commanded gearing completed

Scope	Name	Type	Comment
Output	Busy	BOOL	The function block is not finished
	Active	BOOL	Indicates that the function block has control on the axis
	CommandAborted	BOOL	Command is aborted by another command from other PLCopen function block
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
Inout	Master1	Axis_Ref	Reference to master axis 1
	Master2	Axis_Ref	Reference to master axis 2
	Slave	Axis_Ref	Reference to slave axis

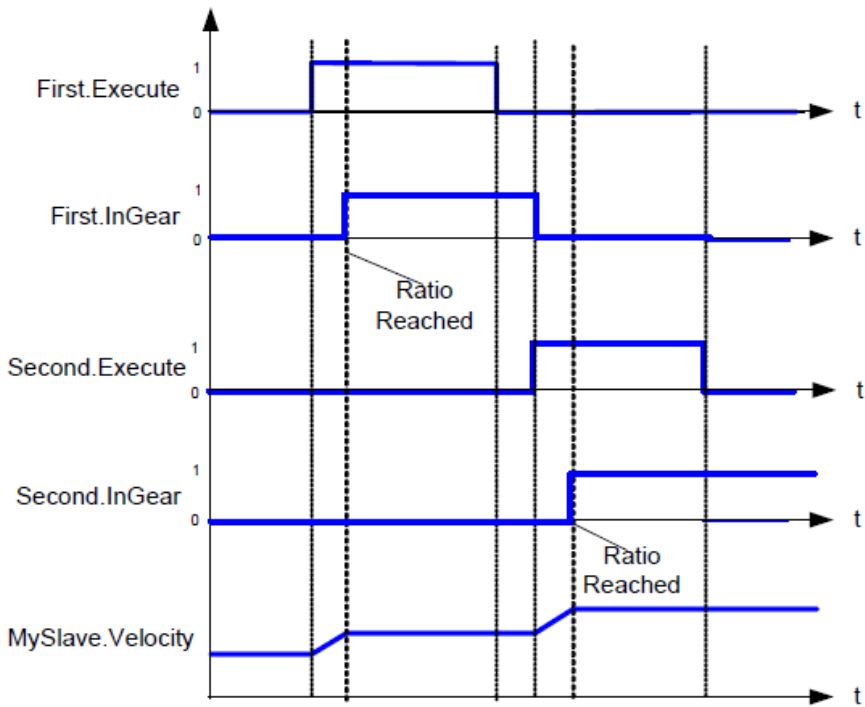
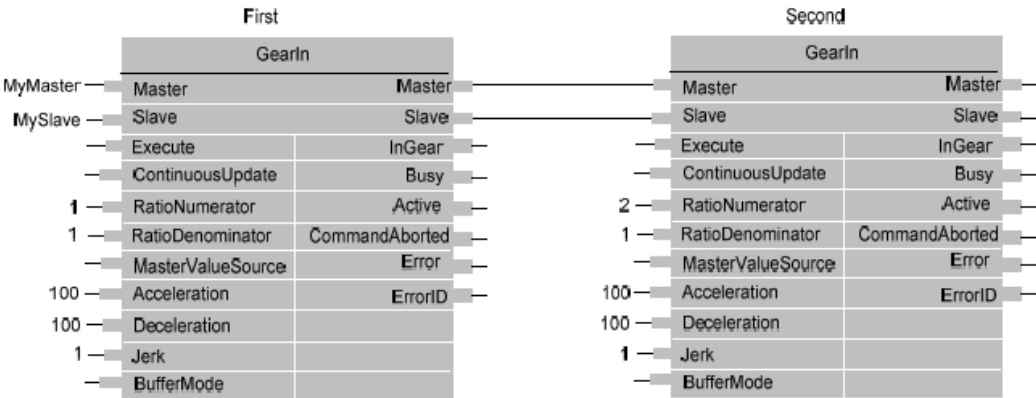
10.2.1.1.4.4 MC_GearIn (FB)

This function block commands a ratio between the velocity of the slave and master axis.



- The slave ramps up to the ratio of the master velocity and locks in when this is reached. Any lost distance during synchronization is not caught up.
- The gearing ratio can be changed while MC_GearIn is running by a rising edge at "Execute" or by using a consecutive MC_GearIn command without the necessity to MC_GearOut first
- InGear is set the first time the ratio is reached.
- After being InGear, a position locking is performed.

Example of GearIn timing diagram



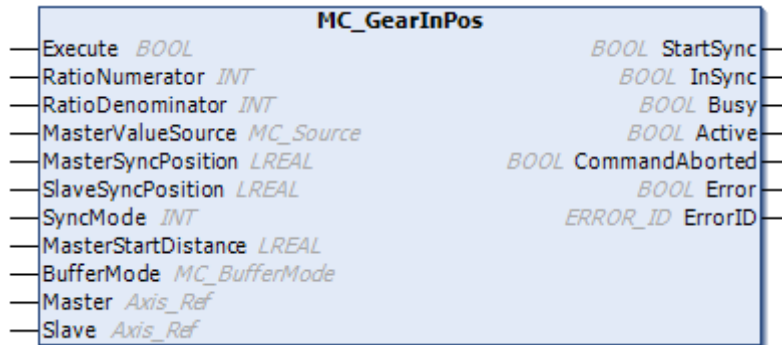
InOut:

Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
	RatioNumerator	INT	Gear ratio numerator, new value is updated only with rising edge of Execute input
	RatioDenominator	INT	Gear ratio denominator, new value is updated only with rising edge of Execute input

Scope	Name	Type	Comment
	MasterValueSource	MC_Source	Decide to use the actual position or reference position of master axis
	Acceleration	LREAL	[u/s ²] Value of the acceleration (increasing energy of the motor), just applied until “insync” is reached. Range: >0
	Deceleration	LREAL	[u/s ²] Value of the deceleration (decreasing energy of the motor), just applied until “insync” is reached. Range: >0
	BufferMode	MC_Buffer-Mode	Not supported, default mcABORTING used
Output	InGear	BOOL	Commanded gearing completed
	Busy	BOOL	The function block is not finished
	Active	BOOL	Indicates that the function block has control on the axis
	CommandAborted	BOOL	Command is aborted by another command from other PLCopen function block
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
Inout	Master	Axis_Ref	Reference to master axis
	Slave	Axis_Ref	Reference to slave axis

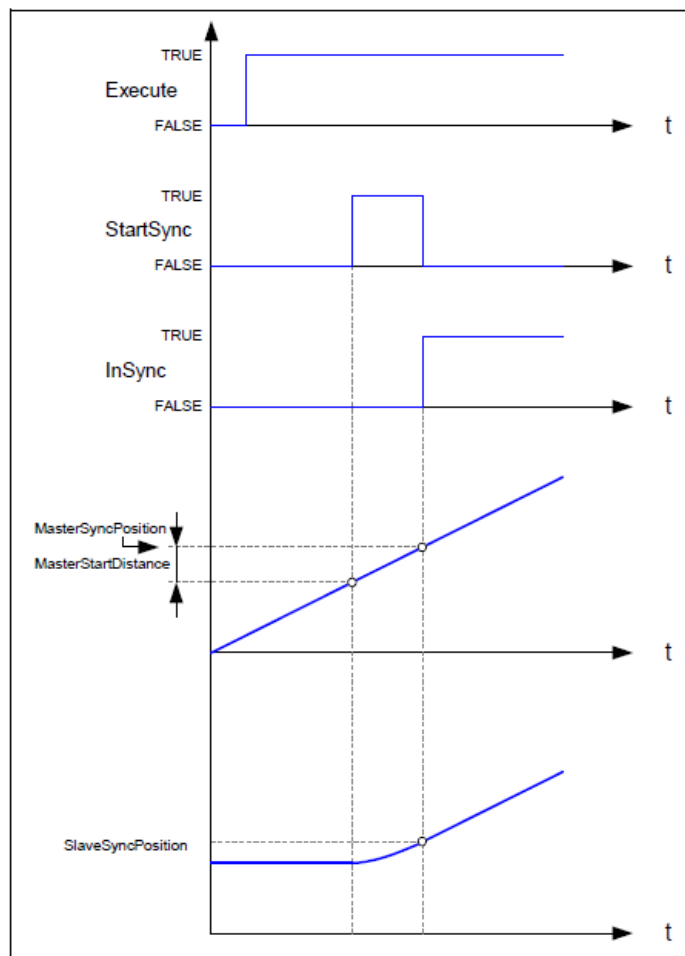
10.2.1.1.4.5 MC_GearInPos (FB)

This function block commands a gear ratio between the position of the slave and master axes from the synchronization point onwards.

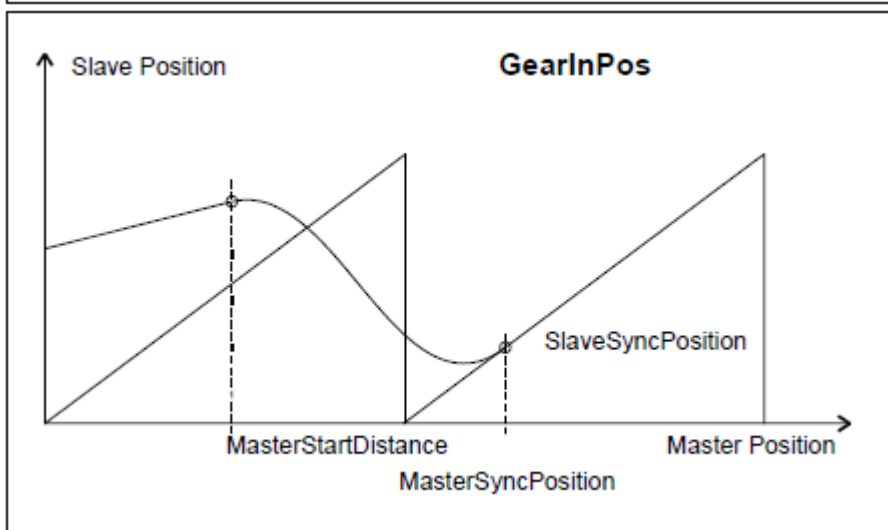
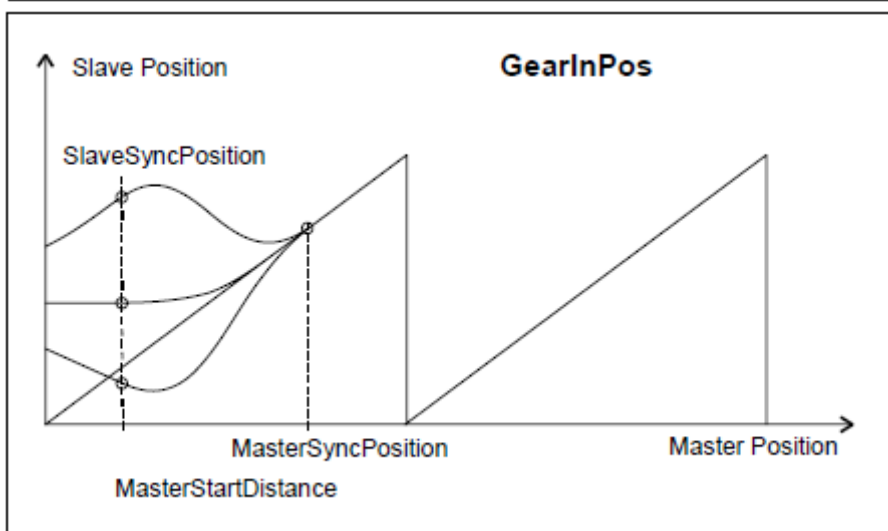
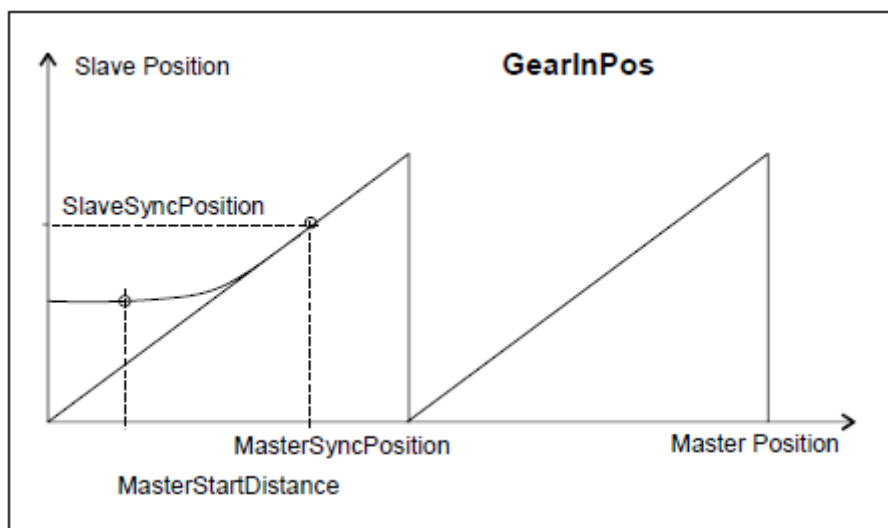


- Any previous motion is continued until master crosses “MasterSyncPosition – MasterStartDistance” in the correct direction (according to the sign of MasterStartDistance).
- At that point in time the output StartSync is set. When a “Stop” command is executed on the “Slave” axis before the synchronization has happened, it inhibits the synchronization and the function block issues “CommandAborted”.
- If the MasterStartDistance is not specified, the system itself could calculate the set point for StartSync based on the other relevant inputs

Example of GearInPos timing diagram



Different examples of MC_GearInPos



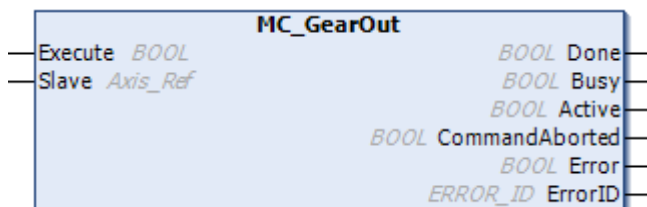
InOut:

Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
	Ratio-Numerator	INT	Gear ratio numerator, new value is updated only with rising edge of Execute input, not while still waiting to start
	Ratio-Denominator	INT	Gear ratio denominator, new value is updated only with rising edge of Execute input, not while still waiting to start
	Master-ValueSource	MC_Source	Decide to use the actual position or reference position of master axis
	MasterSyncPosition	LREAL	The position of the master in the path where the group is insync with the master. (If the 'MasterSyncPosition' does not exist, at the first point of the path the master and slave are synchronized)
	Slave-SyncPosition	LREAL	Slave Position at which the axes are running in sync
	SyncMode	INT	This function block does not support different modes. Synchronization direction is determined by the sign of MasterStartDistance 1 = Sync in matching direction with MasterStartDistance, limit used MasterStartDistance (with respect to actual slave position) to avoid reverse slave direction
	Master-StartDistance	LREAL	The master distance for the slave to start to synchronize to the master. Start synchronizing when the master passes MasterSyncPosition - MasterStartDistance
	Buffer-Mode	MC_BufferMode	Not supported, default mcABORTING used
Output	StartSync	BOOL	Synchronization was started
	InSync	BOOL	Commanded gearing completed and synchronization position passed

Scope	Name	Type	Comment
	Busy	BOOL	The function block is not finished
	Active	BOOL	Indicates that the function block has control on the axis
	CommandAborted	BOOL	Command is aborted by another command from other PLCopen function block
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
Inout	Master	Axis_Ref	Reference to master axis
	Slave	Axis_Ref	Reference to slave axis

10.2.1.1.4.6 MC_GearOut (FB)

This function block disengages the Slave axis from the Master axis



- The slave axis will keep the actual velocity.
- It is assumed that this command is followed by another command, for instance MC_Stop, MC_GearIn, or any other command. If there is no new command, the default condition should be: maintain last velocity.
- After issuing the function block there is no function block active on the slave axis till the next function block is issued (what can result in problems because no motion command is controlling the axis).
- Alternatively, one can read the actual velocity via MC_ReadActualVelocity and issue MC_MoveVelocity on the slave axis with the actual velocity as input. The function block is here because of compatibility reasons



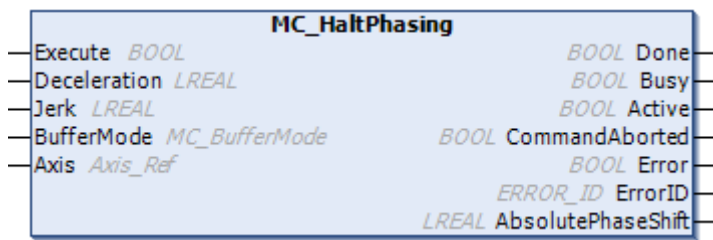
It is not required to use this block to disengage the axis. MC_Stop/MC_Halt or any other command could be used instead.

InOut:

Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
Output	Done	BOOL	Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BOOL	The function block is not finished
	Active	BOOL	Indicates that the function block has control on the axis
	CommandAborted	BOOL	Command is aborted by another command from other PLCOpen function block
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
Inout	Slave	Axis_Ref	Reference to slave axis

10.2.1.1.4.7 MC_HaltPhasing (FB)

This function block commands a controlled motion stop for the phasing movement.



The axis state is not changed



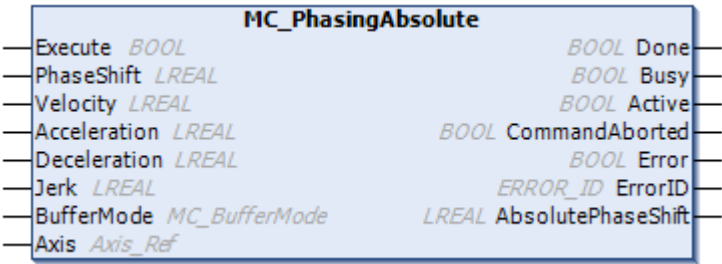
MC_HaltPhasing can be aborted by another phasing command. This function block is applicable for phasing function blocks only.

InOut:


Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
	Deceleration	LREAL	[u/s ²] Value of the deceleration (decreasing energy of the motor). Range: >0. If value = 0, Deceleration will be equal to parameter paraMaxDecelerationAppl
	Jerk	LREAL	[u/s ³] Value of the Jerk. Range: >=0
	BufferMode	MC_Buffer-Mode	Not supported, default mcABORTING used
Out-put	Done	BOOL	Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BOOL	The function block is not finished
	Active	BOOL	Indicates that the function block has control on the axis
	CommandAborted	BOOL	Command is aborted by another command from other PLCopen function block
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
	AbsolutePhase-Shift	LREAL	[u] Actual phase shift of master axis to slave axis, valid while function block is busy
Inout	Axis	Axis_Ref	Reference to axis

10.2.1.1.4.8 MC_PhasingAbsolute (FB)

This function block performs a movement for the relation to the master axis of the specified axis. A real movement is just performed in case the axis is in synchronized motion.

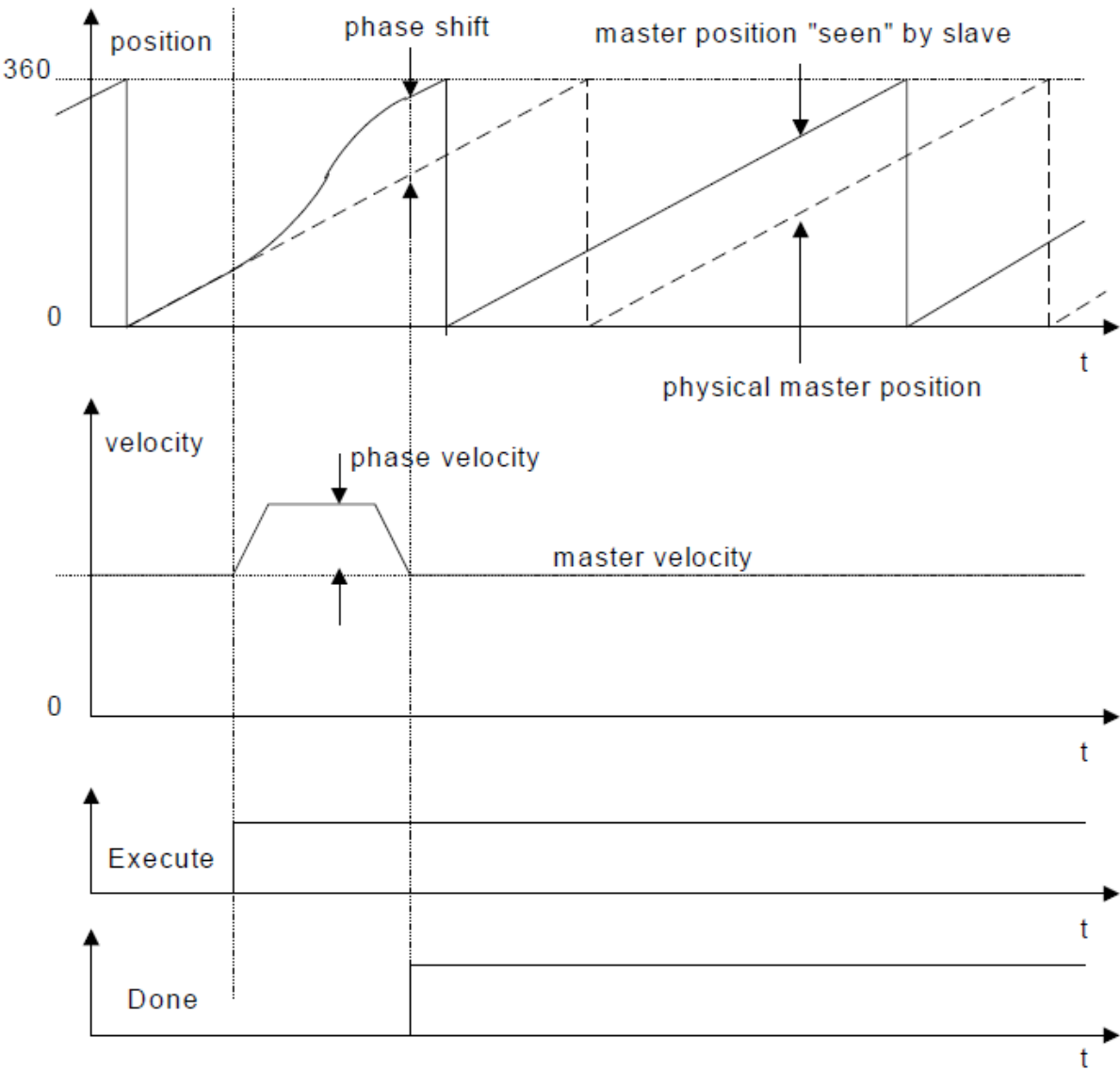


This function block creates an absolute phase shift in the master position of a slave axis. The master position is shifted in relation to the real physical position. This is analogous to opening a coupling on the master shaft for a moment and is used to delay or advance an axis to its master. The phase shift is seen from the slave. The master does not know that there is a phase shift experienced by the slave. The phase shift remains until another “Phasing” command changes it again.

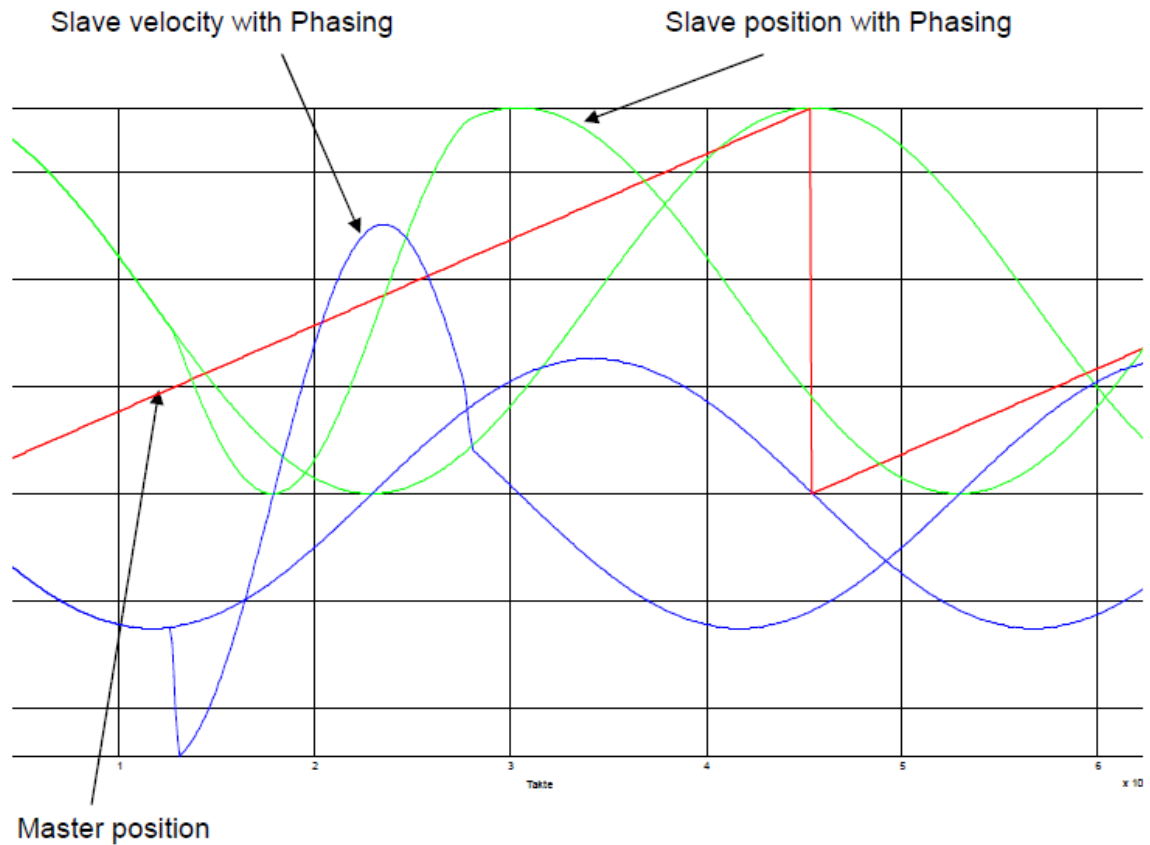


Phase, Velocity, Acceleration, Deceleration and Jerk of a phase shift are controlled by the function block

Timing example of MC_Phasing



The following graph shows the effect of “Phasing”



To halt this function block user must use MC_HaltPhasing function block instead of MC_Stop or MC_Halt

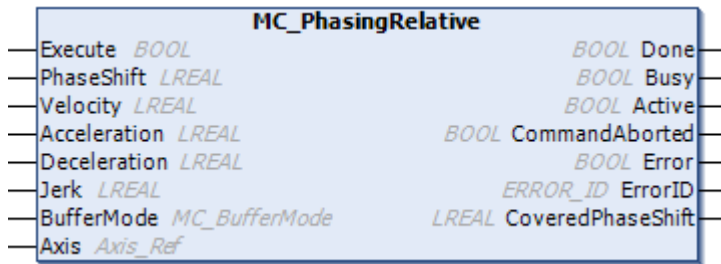
InOut:

Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
	PhaseShift	LREAL	[u] = Technical unit, Absolute phase difference in master
	Velocity	LREAL	[u/s] Value of the maximum velocity (not necessarily reached). Range: >0
	Acceleration	LREAL	[u/s ²] Value of the acceleration (increasing energy of the motor). Range: >0
	Deceleration	LREAL	[u/s ²] Value of the deceleration (decreasing energy of the motor). Range: >0
	Jerk	LREAL	[u/s ³] Value of the jerk. Range: >=0

Scope	Name	Type	Comment
	BufferMode	MC_Buffer-Mode	Not supported, default mcABORTING used
Out-put	Done	BOOL	Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BOOL	The function block is not finished
	Active	BOOL	Indicates that the function block has control on the axis
	CommandAborted	BOOL	Command is aborted by another command from other PLCopen function block
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
	AbsolutePhase-Shift	LREAL	[u] Actual phase shift of master axis to slave axis, valid while function block is busy
Inout	Axis	Axis_Ref	Reference to axis

10.2.1.1.4.9 MC_PhasingRelative (FB)

This function block performs a movement for the relation to the master axis of the specified axis. A real movement is just performed in case the axis is in synchronized motion.

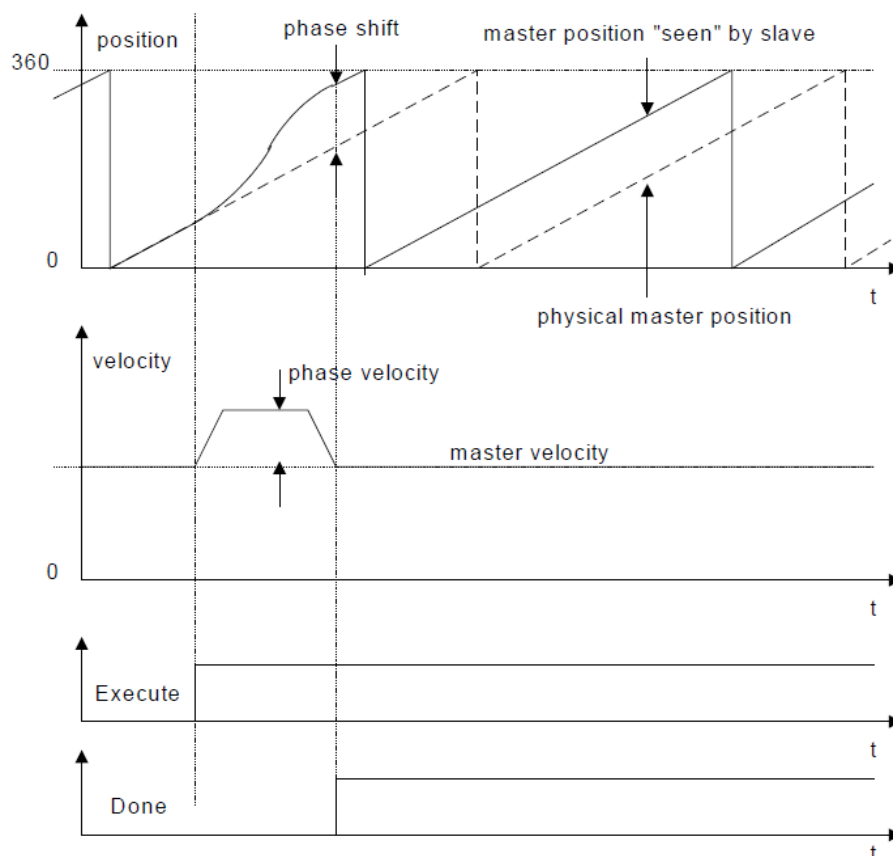


This function block creates a relative phase shift in the master position of a slave axis. The master position is shifted in relation to the real physical position. This is analogous to opening a coupling on the master shaft for a moment and is used to delay or advance an axis to its master. The phase shift is seen from the slave. The master does not know that there is a phase shift experienced by the slave. The phase shift remains until another "Phasing" command changes it again.

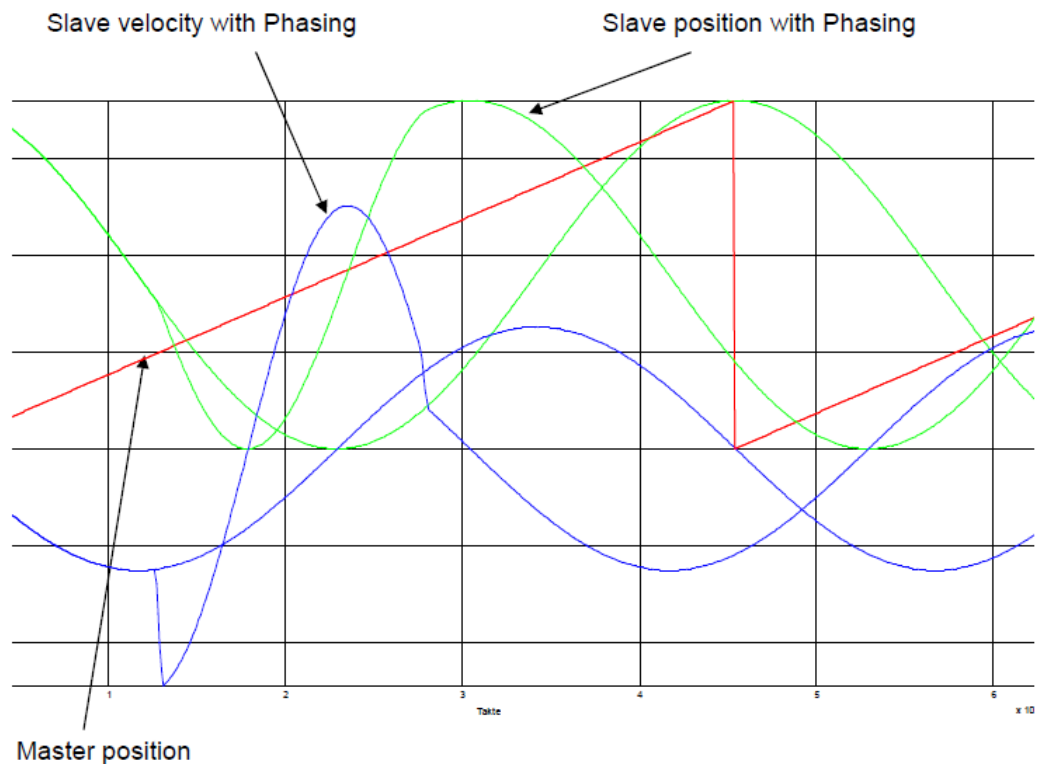


Phase, Velocity, Acceleration, Deceleration and Jerk of a phase shift are controlled by the function block

Timing example of MC_Phasing



The following graph shows the effect of “Phasing”



To halt this function block user must use MC_HaltPhasing function block instead of MC_Stop or MC_Halt

InOut:

Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
	PhaseShift	LREAL	[u] = technical unit, Relative phase difference in master
	Velocity	LREAL	[u/s] Value of the maximum velocity (not necessarily reached). Range: >0
	Acceleration	LREAL	[u/s ²] Value of the acceleration (increasing energy of the motor). Range: >0
	Deceleration	LREAL	[u/s ²] Value of the deceleration (decreasing energy of the motor). Range: >0
	Jerk	LREAL	[u/s ³] Value of the jerk. Range: >=0

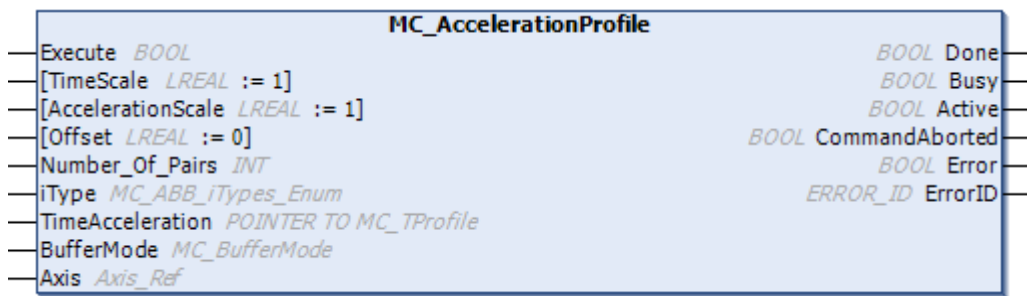
Scope	Name	Type	Comment
	BufferMode	MC_Buffer-Mode	Not supported, default mcABORTING used
Output	Done	BOOL	Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BOOL	The function block is not finished
	Active	BOOL	Indicates that the function block has control on the axis
	CommandAborted	BOOL	Command is aborted by another command from other PLCopen function block
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
	CoveredPhase-Shift	LREAL	Actual phase shift of master axis to slave axis, valid while function block is busy
Inout	Axis	Axis_Ref	Reference to axis

10.2.1.1.5 MC SingleAxis

PLC open motion control single axis function blocks

10.2.1.1.5.1 MC_AccelerationProfile (FB)

This function block commands a time-acceleration locked motion profile.



Alternatively to this function block, the CAM function block coupled to a virtual master can be used.

Example of an acceleration profile

A profile is made from a number of sequential “A to B” positioning points. It is simple to visualize, but requires a lot of sequences for a smooth profile. These requirements are often beyond the capability of low-end servos. Alternatively, by using a modest amount of constant acceleration segments it is possible to define a well-matching motion profile. With this method the capability range of low-end servos can be extended. It is possible to make matching to either:

- Position versus time profile
- Master versus slave axis

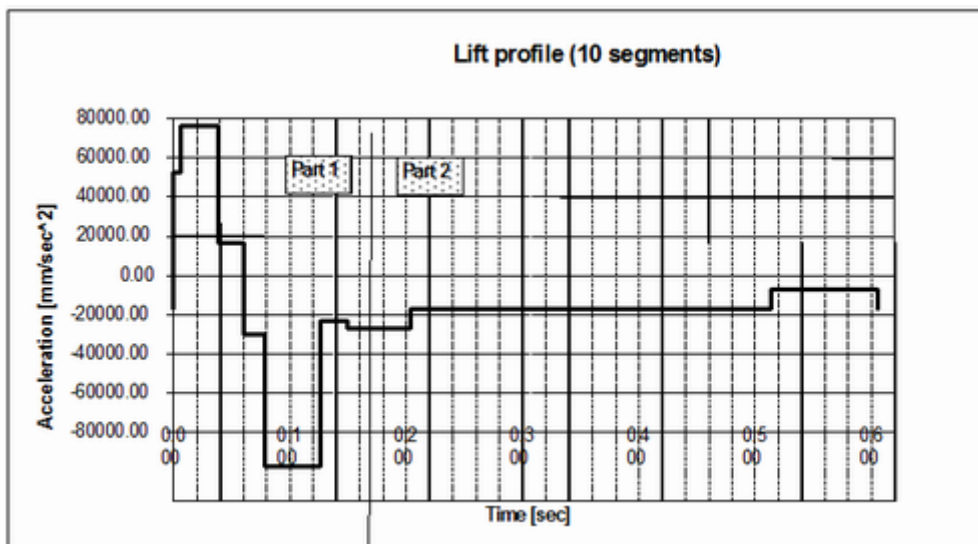
Advantages

- Compact description of a profile.
- Smooth profile properties by nature.
- Low processor power requirements.

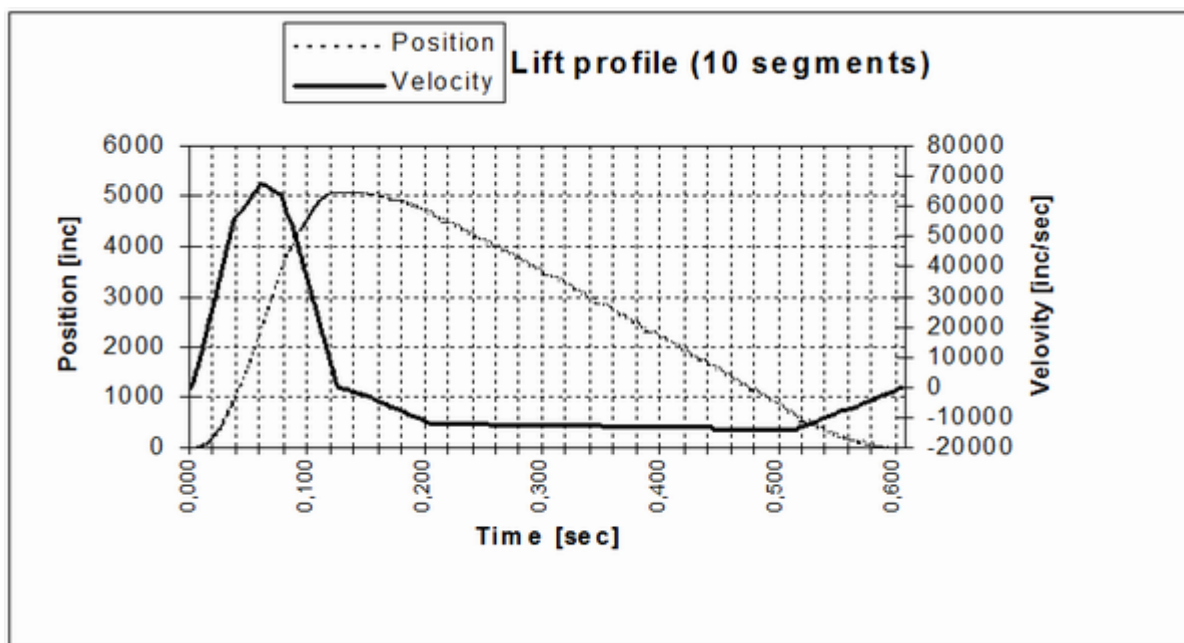
Disadvantages

- Higher programming abstraction level with existing tools.

Acceleration profile, 10 segments only



Resulting position profile



MC_TProfile is an ABB specific datatype.



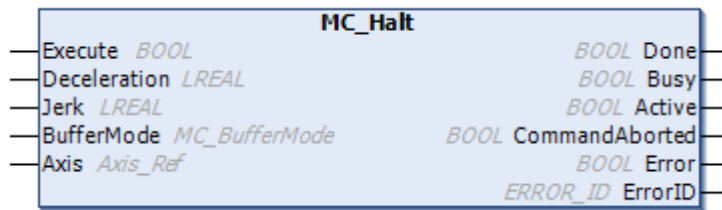
When Done = TRUE (profile is completed), Axis will run with the last Velocity value.

InOut:

Scope	Name	Type	Initial	Comment
Input	Execute	BOOL		Starts the function block at rising edge
	TimeScale	LREAL	1	Overall time scaling factor of the profile. Range: >0
	AccelerationScale	LREAL	1	Scale factor for acceleration amplitude. Range: AccelerationScale <> 0
	Offset	LREAL	0	Overall offset for profile, the profile result will be increased by Offset
	Number_Of_Pairs	INT		Number of sampling points, elements in TimeAcceleration array. Range: >=2
	iType	MC_ABB_iTypes_Enum		Type of interpolation. Possible values are: MCA_SPLINE_COMPLETE MCA_SPLINE_NATURAL MCA_POLY5 MCA_POLY3 MCA_LINEAR
	TimeAcceleration	POINTER TO MC_TProfile		Reference to Time / Acceleration
	BufferMode	MC_BufferMode		Not supported, default mcABORTING used
Output	Done	BOOL		Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BOOL		The function block is not finished
	Active	BOOL		Indicates that the function block has control on the axis
	CommandAborted	BOOL		Command is aborted by another command from other PLCopen function block
	Error	BOOL		Signals that error has occurred within function block
	ErrorID	ERROR_ID		Error identification. For error details refer to Enumeration ERROR_ID
Inout	Axis	Axis_Ref		Reference to axis

10.2.1.1.5.2 MC_Halt (FB)

This function block commands a controlled motion stop.



The axis is moved to the state “DiscreteMotion”, until the velocity is zero. With the Done output set, the state is transferred to “STANDSTILL”.



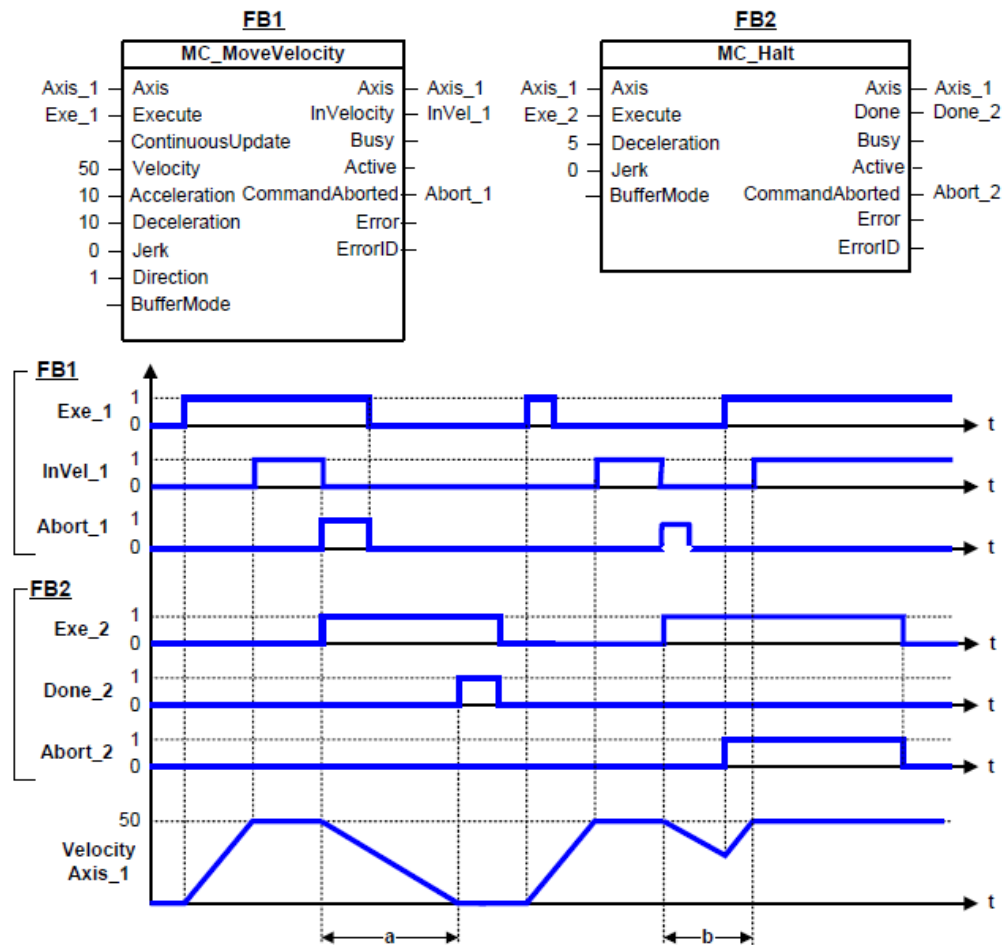
MC_Halt can be aborted by another command.

- MC_Halt is used to stop the axis under normal operation conditions. In non-buffered mode it is possible to set another motion command during deceleration of the axis, which will abort the MC_Halt and will be executed immediately.
- If this command is active the next command can be issued. E.g. a driverless vehicle detects an obstacle and needs to stop. MC_Halt is issued. Before the standstill is reached the obstacle is removed and the motion can be continued by setting another motion command, so the vehicle does not stop.

Example

Behavior of MC_Halt in combination with MC_MoveVelocity

- A rotating axis is ramped down with Function Block MC_Halt.
- Another motion command overrides the MC_Halt command. MC_Halt allows this, in contrast to MC_Stop. The axis can accelerate again without reaching standstill.

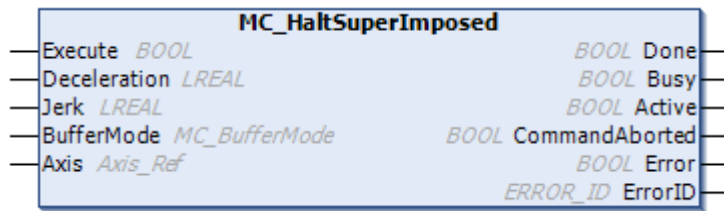
**InOut:**

Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
	Deceleration	LREAL	[u/s ²] Value of the deceleration (decreasing energy of the motor). Range: >0. If value = 0, Deceleration will be equal to parameter paraMaxDecelerationAppl
	Jerk	LREAL	[u/s ³] Value of the Jerk. Range: >=0
	BufferMode	MC_Buffer-Mode	not supported, default mcABORTING used
Out-put	Done	BOOL	Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BOOL	The function block is not finished

Scope	Name	Type	Comment
	Active	BOOL	Indicates that the function block has control on the axis
	CommandAborted	BOOL	Command is aborted by another command from other PLCopen function block
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
Inout	Axis	Axis_Ref	Reference to axis

10.2.1.1.5.3 MC_HaltSuperImposed (FB)

This function block commands a halt to all superimposed motions of the axis. The underlying motion is not interrupted.

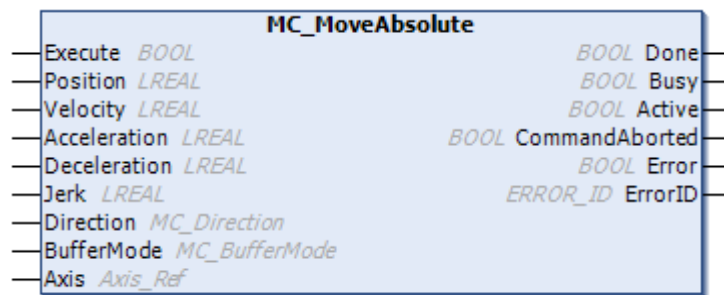


InOut:

Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
	Deceleration	LREAL	[u/s ^{°2}] Value of the deceleration (decreasing energy of the motor). Range: >0. If value = 0, Deceleration will be equal to parameter paraMaxDecelerationAppl
	Jerk	LREAL	[u/s ^{°°3}] Value of the jerk. Range: >=0
	BufferMode	MC_Buffer-Mode	Not supported, default mcABORTING used
Output	Done	BOOL	Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BOOL	The function block is not finished and new output values are to be expected
	Active	BOOL	Indicates that the function block has control on the axis
	CommandAborted	BOOL	Command is aborted by another command from other PLCopen function block
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
Inout	Axis	Axis_Ref	Reference to axis

10.2.1.1.5.4 MC_MoveAbsolute (FB)

This function block commands a controlled motion to a specified absolute position.



- This action completes with velocity zero if no further action are pending.
- If there is only one mathematical solution to reach the commanded position (like in linear systems), the value of the input direction is ignored.
- For modulo axis, valid absolute position values are in the range of [0, [360, (360 is excluded), or corresponding range. The application, however, may shift the commanded position of MC_MoveAbsolute into the corresponding modulo range. For relative positions, modulo 360 is applicable.
- The Enum type “shortest_way” is focused to a trajectory which will go through the shortest route. The decision which direction to go is based on the current position where the command is issued.

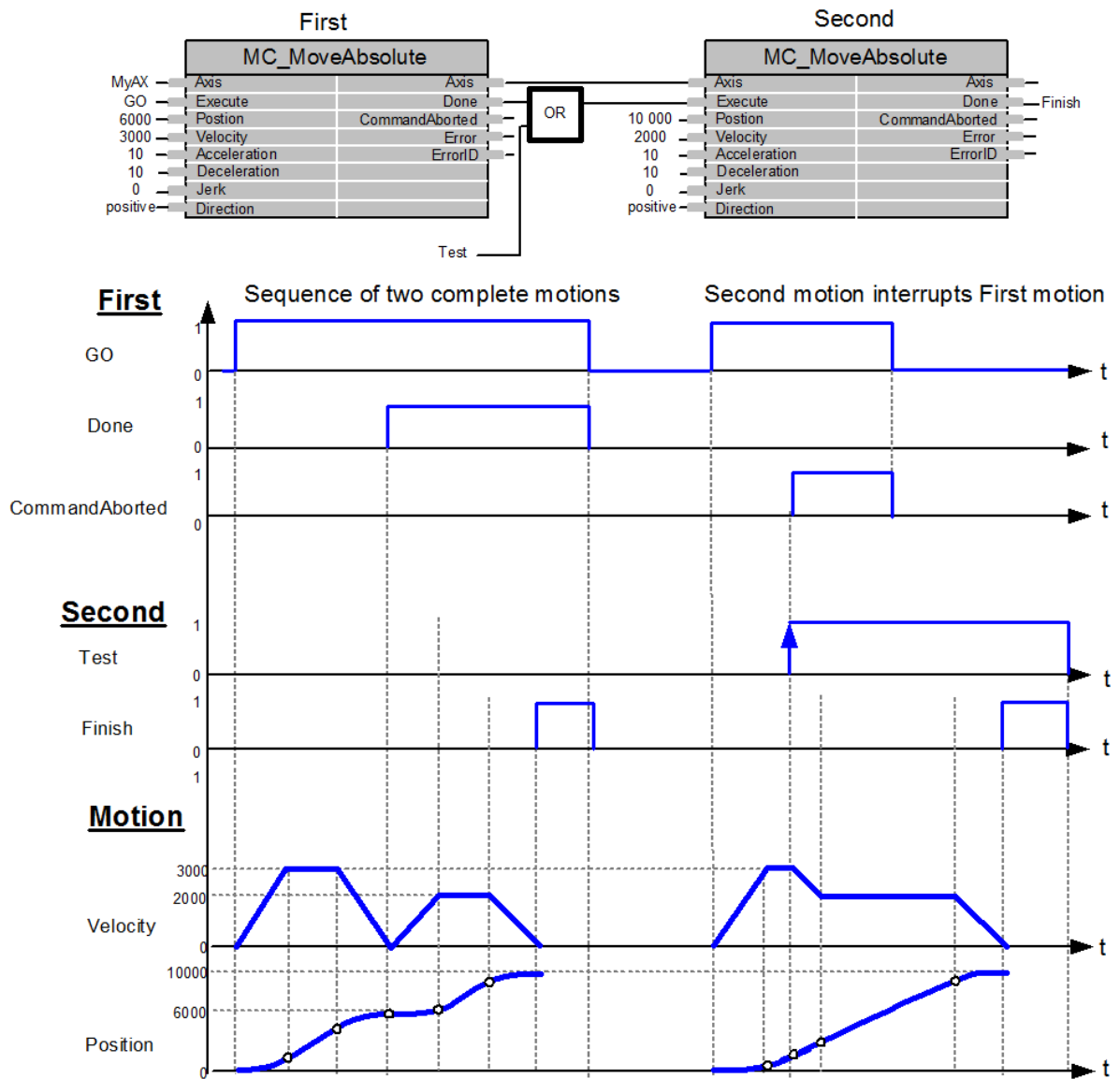


With every rising edge at “Execute”, modified input parameters will be used.

Example

The following figure shows two examples of the combination of two function blocks MC_MoveAbsolute.

- The left part of timing diagram illustrates the case if the second function block is called after the first one. If first reaches the commanded position of 6000 (and the velocity is 0) then the output Done causes the second function block to move to the position 10000.
- The right part of the timing diagram illustrates the case if the second move function block starts the execution while the first function block is still executing. In this case the first motion is interrupted and aborted by the test signal during the constant velocity of the first Function Block. The second function block moves directly to the position 10000 although the position of 6000 is not yet reached



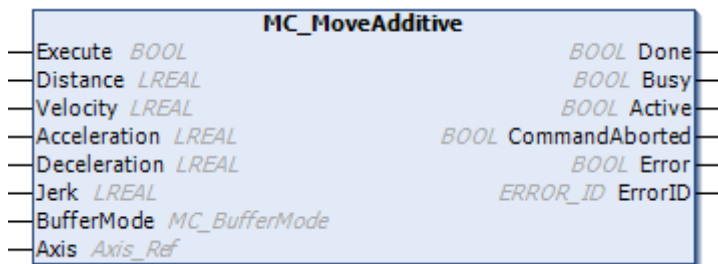
Input validation is done at the rising edge of Execute. If function block is in Active/Busy state, new value at input will not be validated. If value passed is invalid function block will continue execution with last valid value.

InOut:

Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
	Position	LREAL	[u] Reference position
	Velocity	LREAL	[u/s] Value of the maximum velocity (not necessarily reached). Range: >0
	Acceleration	LREAL	[u/s ²] Value of the acceleration (increasing energy of the motor). Range: >0. If value = 0, Acceleration will be equal to parameter para-MaxAccelerationAppl
	Deceleration	LREAL	[u/s ²] Value of the deceleration (decreasing energy of the motor). Range: >0. If value = 0, Deceleration will be equal to parameter para-MaxDecelerationAppl
	Jerk	LREAL	[u/s ³] Value of the Jerk. Range: >=0
	Direction	MC_Direction	Positive, shortest, negative, current, positive_stop, negative_stop, Current_Stop
	BufferMode	MC_Buffer-Mode	Not supported, default mcABORTING used
Output	Done	BOOL	Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BOOL	The function block is not finished
	Active	BOOL	Indicates that the function block has control on the axis
	CommandAborted	BOOL	Command is aborted by another command from other PLCOpen function block
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
Inout	Axis	Axis_Ref	Reference to axis

10.2.1.1.5.5 MC_MoveAdditive (FB)

This function block commands a controlled motion of a specified relative distance additional to the most recent commanded position in the discrete motion state.

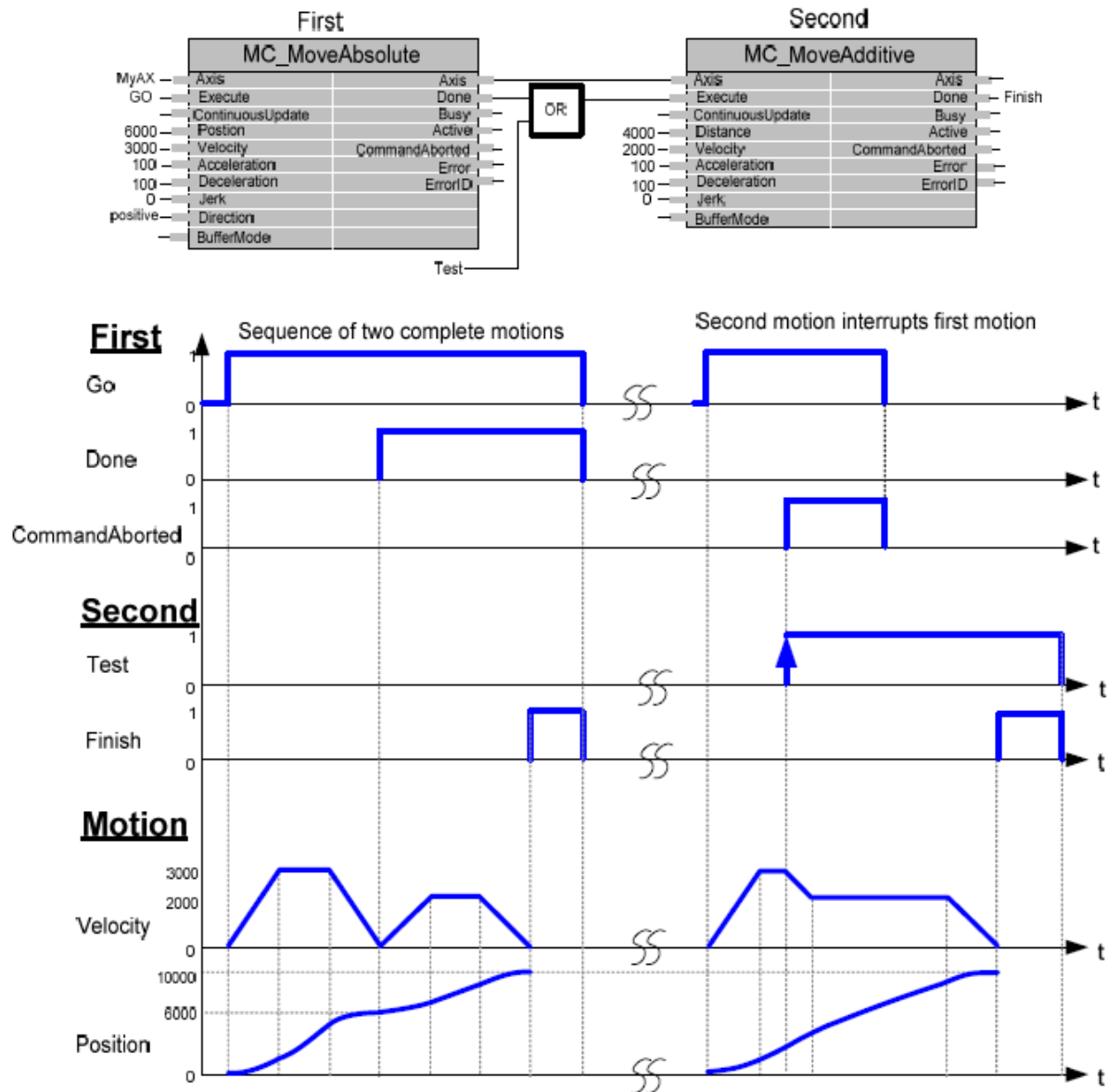


The most recent commanded position may be the result of a previous MC_MoveAdditive motion which was aborted. If the function block is activated in the Continuous Mode the specified relative distance is added to the actual position at the time of the execution.

Example

Examples of the combination of two function blocks while the axis is in state Discrete Motion

- The left part of timing diagram illustrates the case if the second function block is called after the first one. If the first one reaches the commanded distance 6000 (and the velocity is 0) then the output “Done” causes the second function block to move to the distance 10000.
- The right part of the timing diagram illustrates the case if the second move function blocks starts the execution while the first function block is still executing. In this case the first motion is interrupted and aborted by the test signal during the constant velocity of the first function block. The second function block adds on the previous commanded position of 6000 the distance 4000 and moves the axis to the resulting position of 10000

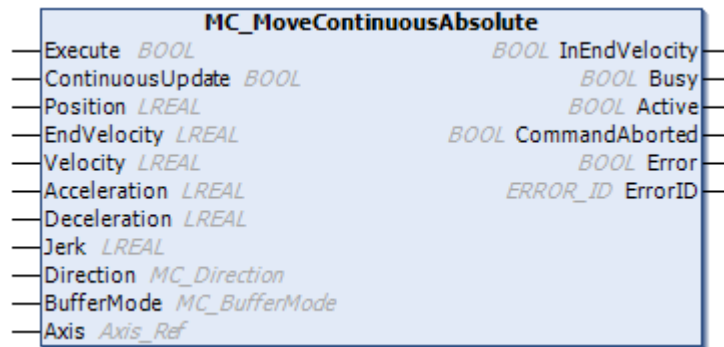


InOut:

Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
	Distance	LREAL	[u] = Technical unit, Relative distance for the motion
	Velocity	LREAL	[u/s] Value of the maximum velocity (not necessarily reached). Range: >0
	Acceleration	LREAL	[u/s ²] Value of the acceleration (increasing energy of the motor). Range: >0. If value = 0, Acceleration will be equal to parameter paraMaxAccelerationAppl
	Deceleration	LREAL	[u/s ²] Value of the deceleration (decreasing energy of the motor). Range: >0. If value = 0, Deceleration will be equal to parameter paraMaxDecelerationAppl
	Jerk	LREAL	[u/s ³] Value of the Jerk. Range: >=0
	BufferMode	MC_Buffer-Mode	Not supported, default mcABORTING used
Out-put	Done	BOOL	Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BOOL	The function block is not finished
	Active	BOOL	Indicates that the function block has control on the axis
	CommandAborted	BOOL	Command is aborted by another command from other PLCopen function block
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
Inout	Axis	Axis_Ref	Reference to axis

10.2.1.1.5.6 MC_MoveContinuousAbsolute (FB)

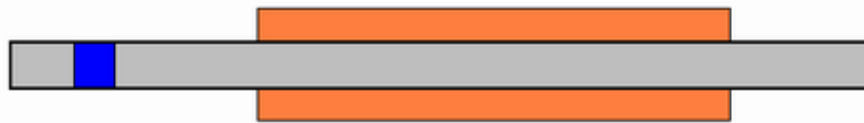
This function block commands a controlled motion to a specified absolute position ending with the specified velocity.



- If the commanded position is reached and no new motion command is put into the buffer, the axis continues to run with the specified “EndVelocity”.
- The function block will start the axis with state DiscreteMotion, while positioning.
- It will change to state Continuous Motion (meaning: it will not stop by itself) with EndVelocity $\neq 0$.
- It will change to standstill with EndVelocity = 0.

One use case for MC_MoveContinuousAbsolute is a linear cutter. One linear axis that is carrying a laser device that is used to cut a workpiece

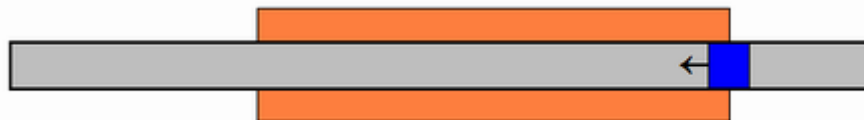
Start from IrIdlePos.



1. Move the laser with fast velocity over the position IrStartCutPos. The laser is off during this movement



2. Turn back and make sure to have the speed IrCutVelocity when at IrStartCutPos. At this position, switch the laser on



3. Travel over the work piece with this constant speed while the laser is on



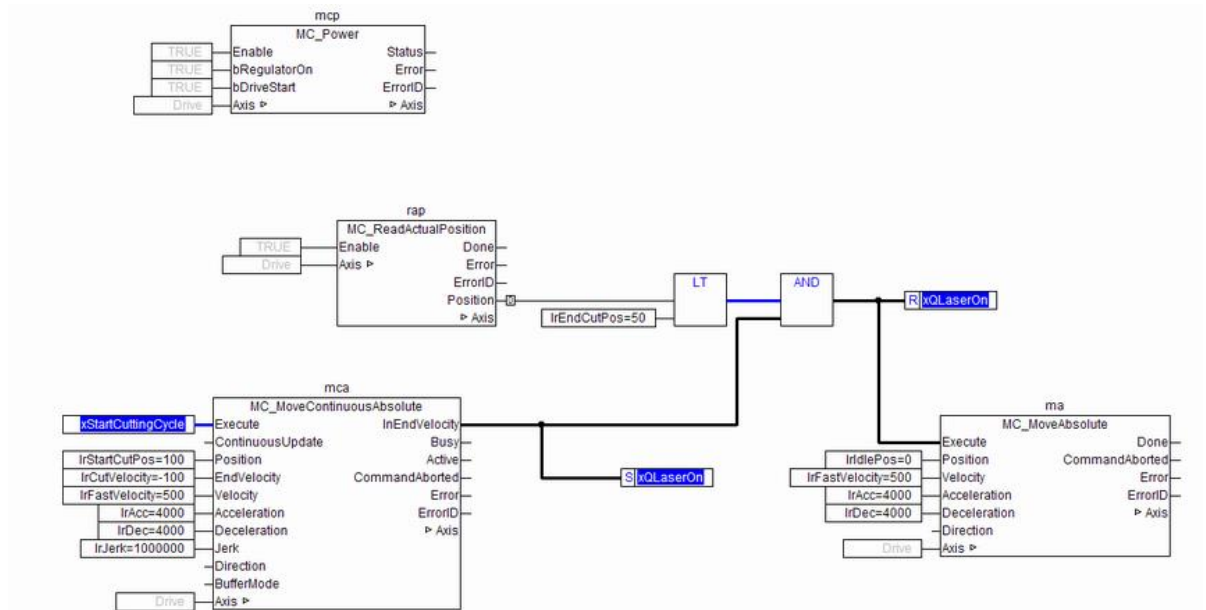
4. When reaching IrEndCutPos switch off the laser and move back to idle position with fast velocity



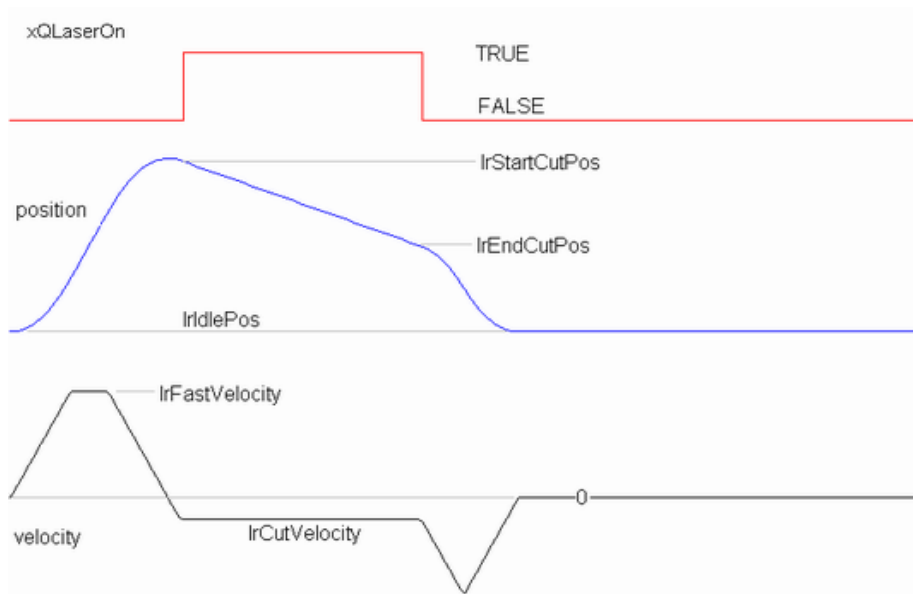
During the cutting process the laser must be moved with a fix velocity, no acceleration or deceleration phase can be tolerated. The laser must be moved to its waiting position after the cutting is done

Example

The explained movement can be achieved with the function block MC_MoveContinuousAbsolute in the following way:



Started with a rising edge of xStartCuttingCycle, the instance “mca” of MC_MoveContinuousAbsolute will move the axis with “IrFastVelocity” over “IrStartCutPos”, turn back and have the speed “IrCutVelocity” when reaching “IrStartCutPos” again in negative direction. In this point in time, “InEndVelocity” is set and the laser is switched on. As no other motion function block interrupts this movement, MC_MoveContinuousAbsolute will keep travelling in negative direction with the current speed. After the axis has overstepped the position “IrEndPos”, where the laser is switched off, the MC_MoveAbsolute instance “ma” moves the axis with high speed to its idle position



Input validation is done at the rising edge of Execute. If function block is in Active/Busy state, new value at input will not be validated. If value passed is invalid function block will continue execution with last valid value.

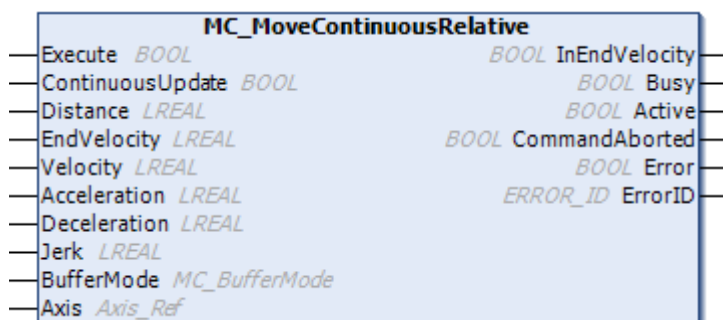
InOut:

Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
	ContinuousUpdate	BOOL	Decide if new input parameters are processed during the movement
	Position	LREAL	[u] Reference position
	EndVelocity	LREAL	[u/s] Signed value for the end velocity, determines the direction when ending the positioning movement
	Velocity	LREAL	[u/s] Value of the maximum velocity (not necessarily reached). Range: >0
	Acceleration	LREAL	[u/s ²] Value of the acceleration (increasing energy of the motor). Range: >0, If value = 0, Acceleration will be equal to parameter <code>paraMaxAccelerationAppl</code>
	Deceleration	LREAL	[u/s ²] Value of the deceleration (decreasing energy of the motor). Range: >0, If value = 0, Deceleration will be equal to parameter <code>paraMaxDecelerationAppl</code>

Scope	Name	Type	Comment
	Jerk	LREAL	[u/s°3] Value of the jerk. Range: >=0
	Direction	MC_Direction	Positive, Shortest, Negative, Current
	BufferMode	MC_Buffer-Mode	Not supported, default mcABORTING used
Out-put	InEndVelocity	BOOL	Commanded position finally reached
	Busy	BOOL	The function block is not finished
	Active	BOOL	Indicates that the function block has control on the axis
	CommandAborted	BOOL	Command is aborted by another command from other PLCopen function block
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
Inout	Axis	Axis_Ref	Reference to axis

10.2.1.1.5.7 MC_MoveContinuousRelative (FB)

This function block commands a controlled motion of a specified relative distance, ending with the specified velocity.

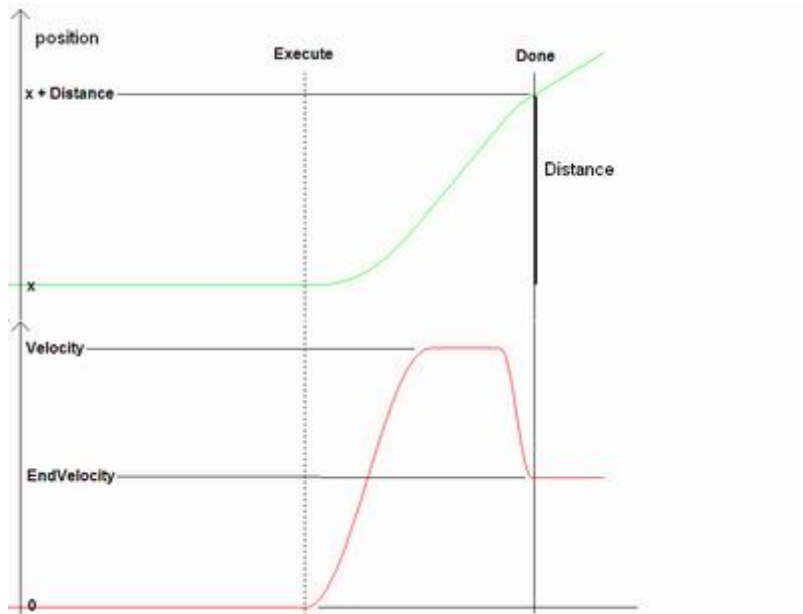


- If the commanded position is reached and no new motion command is put into the buffer, the axis continues to run with the specified “EndVelocity”.
- The function block will start the axis with state DiscreteMotion, while positioning.
- It will change to state continuous motion (meaning: it will not stop by itself) with EndVelocity <> 0.

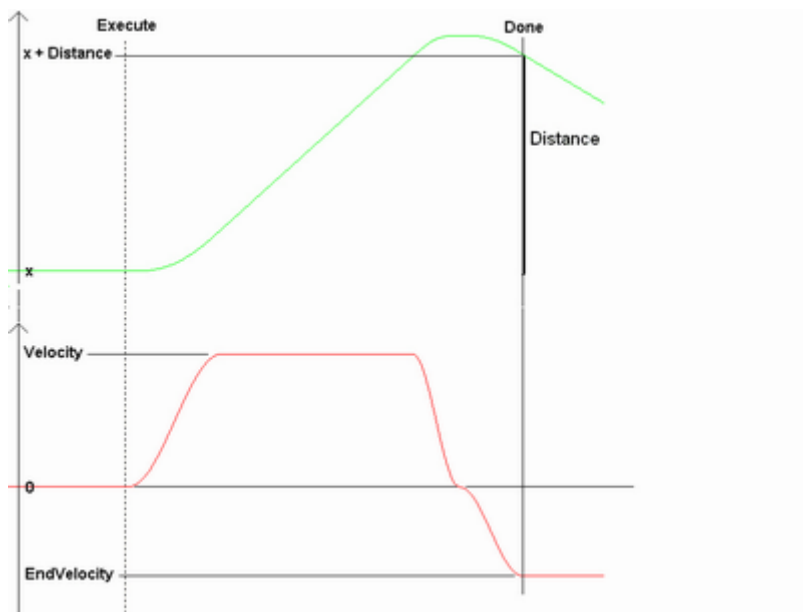
- It will change to standstill with EndVelocity = 0.
- This function block is specified here for systems without the support for the "BufferMode".

Sampling traces showing the effect of the sign of the value of the input EndVelocity

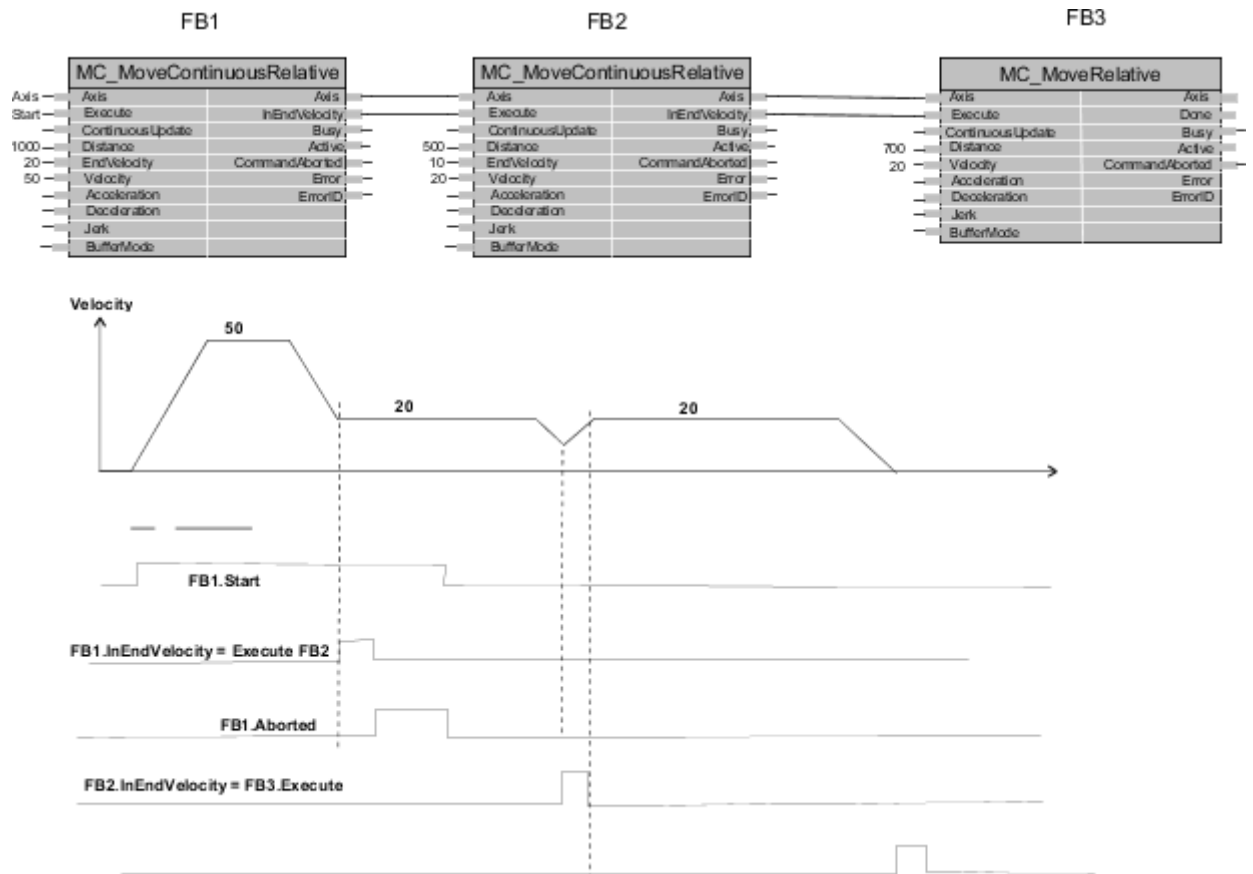
Input EndVelocity with positive direction



Input EndVelocity with negative direction



Example for MC_MoveContinuousRelative



Input validation is done at the rising edge of Execute. If function block is in Active/Busy state, new value at input will not be validated. If value passed is invalid function block will continue execution with last valid value.

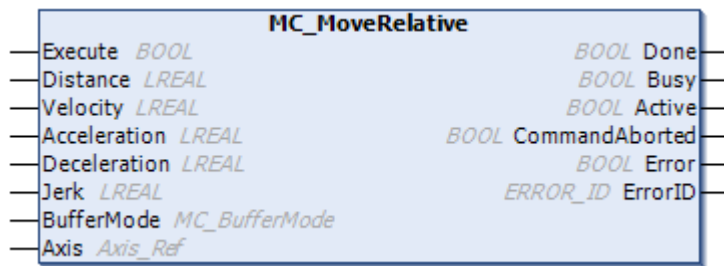
InOut:

Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
	ContinuousUpdate	BOOL	Decide if new input parameters are processed during the movement
	Distance	LREAL	[u] = Technical unit, Relative distance for the motion
	EndVelocity	LREAL	[u/s] Signed value for the end velocity, determines the direction when ending the positioning movement
	Velocity	LREAL	[u/s] Value of the maximum velocity (not necessarily reached). Range: >0

Scope	Name	Type	Comment
	Acceleration	LREAL	[u/s ²] Value of the acceleration (increasing energy of the motor). Range: >0, If value = 0, Acceleration will be equal to parameter paraMaxAccelerationAppl
	Deceleration	LREAL	[u/s ²] Value of the deceleration (decreasing energy of the motor). Range: >0, If value = 0, Deceleration will be equal to parameter paraMaxDecelerationAppl
	Jerk	LREAL	[u/s ³] Value of the jerk. Range: >=0
	BufferMode	MC_Buffer-Mode	Not supported, default mcABORTING used
Out-put	InEndVelocity	BOOL	Commanded position finally reached
	Busy	BOOL	The function block is not finished
	Active	BOOL	Indicates that the function block has control on the axis
	CommandAborted	BOOL	Command is aborted by another command from other PLCopen function block
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
Inout	Axis	Axis_Ref	Reference to axis

10.2.1.1.5.8 MC_MoveRelative (FB)

This function block commands a controlled motion of a specified distance relative to the actual position at the time of the execution.

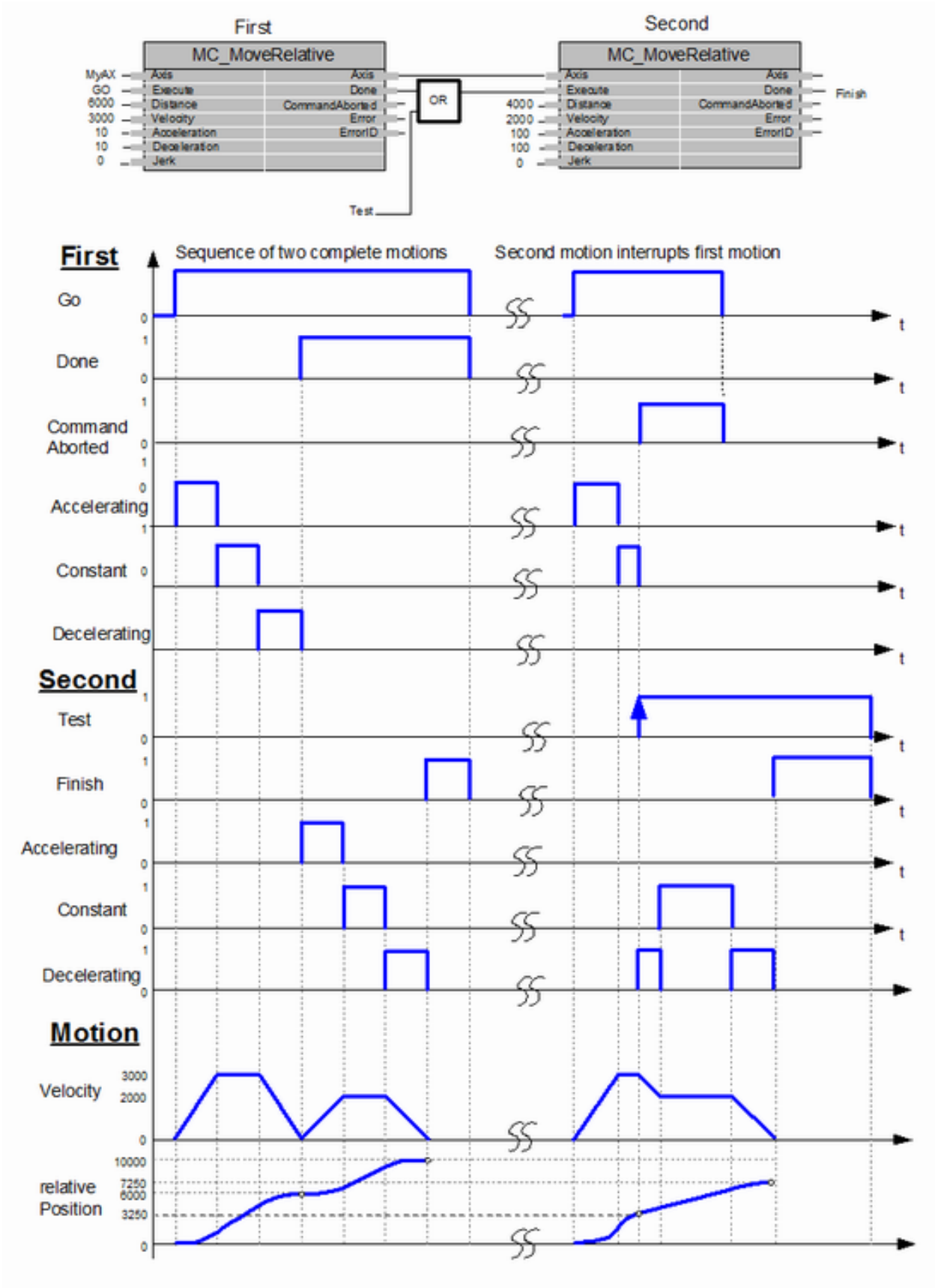


- This action completes with velocity zero if no further action are pending.

Example

The following figure shows the example of the combination of two MC_MoveRelative function blocks:

- The left part of timing diagram illustrates the case if the second function block is called after the first one. If the first one reaches the commanded distance 6000 (and the velocity is 0) then the output Done causes the second Function Block to move to the distance 10000.
- The right part of the timing diagram illustrates the case if the second move function blocks starts the execution while the first function block is still executing. In this case the first motion is interrupted and aborted by the test signal during the constant velocity of the first function block. The second function block adds on the actual position of 3250 the distance 4000 and moves the axis to the resulting position of 7250.



InOut:

Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
	Distance	LREAL	[u] = Technical unit, Relative distance for the motion
	Velocity	LREAL	[u/s] Value of the maximum velocity (not necessarily reached). Range: >0
	Acceleration	LREAL	[u/s ²] Value of the acceleration (increasing energy of the motor). Range: >0. If value = 0, Acceleration will be equal to parameter paraMaxAccelerationAppl
	Deceleration	LREAL	[u/s ²] Value of the deceleration (decreasing energy of the motor). Range: >0. If value = 0, Deceleration will be equal to parameter paraMaxDecelerationAppl
	Jerk	LREAL	[u/s ³] Value of the jerk. Range: >=0
	BufferMode	MC_Buffer-Mode	Not supported, default mcABORTING used
Out-put	Done	BOOL	Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BOOL	The function block is not finished
	Active	BOOL	Indicates that the function block has control on the axis
	CommandAborted	BOOL	Command is aborted by another command from other PLCopen function block
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
Inout	Axis	Axis_Ref	Reference to axis

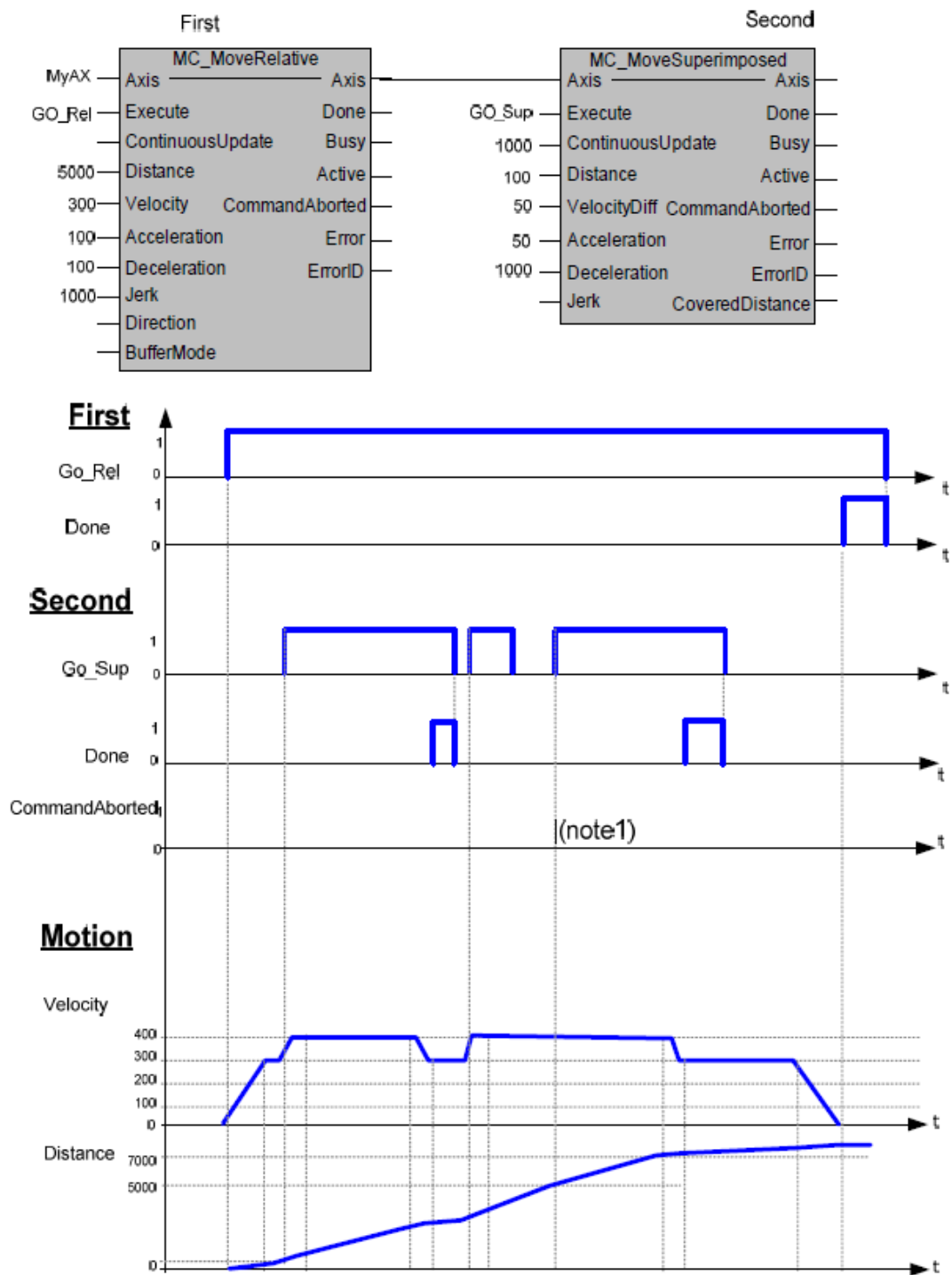
10.2.1.1.5.9 MC_MoveSuperImposed (FB)

This function block commands a controlled motion of a specified relative distance additional to an existing motion. The existing Motion is not interrupted, but is superimposed by the additional motion.



- If MC_MoveSuperImposed is active, then any other command in aborting mode except MC_MoveSuperImposed will abort both motion commands: Both the MC_MoveSuperImposed and the underlying motion command. In any other mode, the underlying motion command is not aborted
- If MC_MoveSuperImposed is active and another MC_MoveSuperImposed is commanded, only the ongoing MC_MoveSuperImposed command is aborted, and replaced by the new MC_MoveSuperImposed, but not the underlying motion command
- The function block MC_MoveSuperimposed causes a change of the velocity, if applicable, the commanded position of an ongoing motion in all relevant states
- In the state StandStill the function block MC_MoveSuperimposed acts like MC_MoveRelative.
- The values of Acceleration, Deceleration and Jerk are additional values to the ongoing motion, not absolute ones. With this, the underlying function block always finishes its job in the same period of time regardless of whether a MC_MoveSuperimposed function block takes place concurrently
- MC_MoveSuperimposed acts on the slave axis, while MC_Phasing acts on the master side, as seen from the slave.

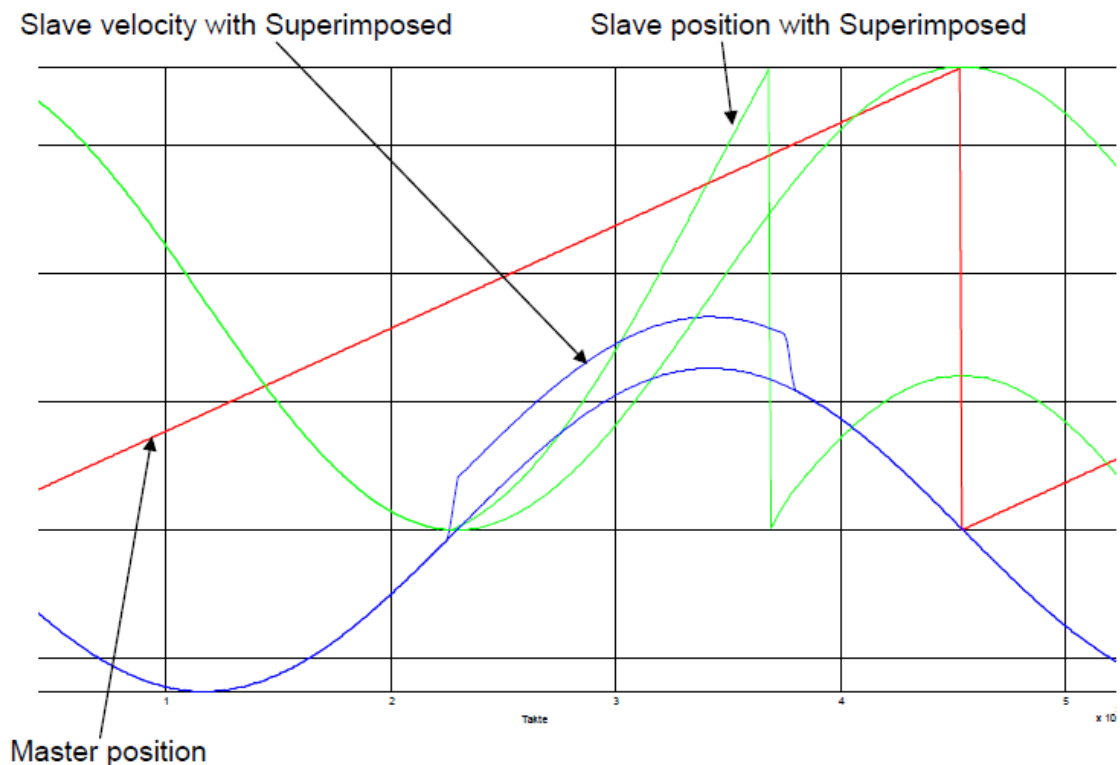
Timing diagram of MC_MoveSuperImposed



- The CommandAborted is not visible here, because the new command works on the same instance
- The end position is between 7000 and 8000, depending on the timing of the aborting of the second command set for the MC_MoveSuperImposed.

Example of MC_MoveSuperimposed during Camming with modulo axes

In green color the slave position is shown both with and without MC_MoveSuperimposed



At Slave velocity, the double line shows the effect of MoveSuperimposed while in Synchronized Motion during Camming. The same is valid for the related slave position.

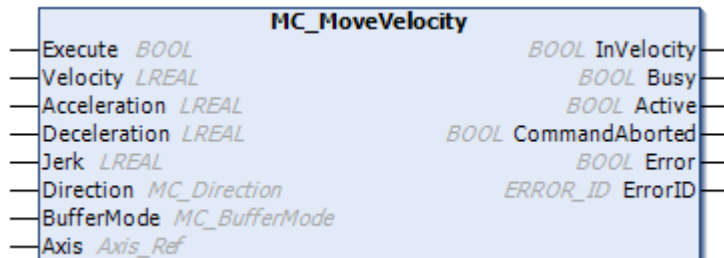
InOut:

Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
	Distance	LREAL	[u] = technical unit, Relative distance for the motion
	Velocity-Diff	LREAL	[u/s] Value of the maximum velocity difference to the ongoing motion (not necessarily reached)
	Acceleration	LREAL	[u/s ²] Value of the acceleration (increasing energy of the motor). Range: >0. If value = 0, Acceleration will be equal to parameter paraMaxAccelerationAppl
	Deceleration	LREAL	[u/s ²] Value of the deceleration (decreasing energy of the motor). Range: >0. If value = 0, Deceleration will be equal to parameter paraMaxDecelerationAppl
	Jerk	LREAL	[u/s ³] Value of the jerk. Range: >=0

Scope	Name	Type	Comment
Out-put	Done	BOOL	Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BOOL	The function block is not finished
	Active	BOOL	Indicates that the function block has control on the axis
	CommandAborted	BOOL	Command is aborted by another command from other PLCopen function block
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ER-ROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
Inout	Axis	Axis_Ref	Reference to axis

10.2.1.1.5.10 MC_MoveVelocity (FB)

This function block commands a never ending controlled motion at a specified velocity.

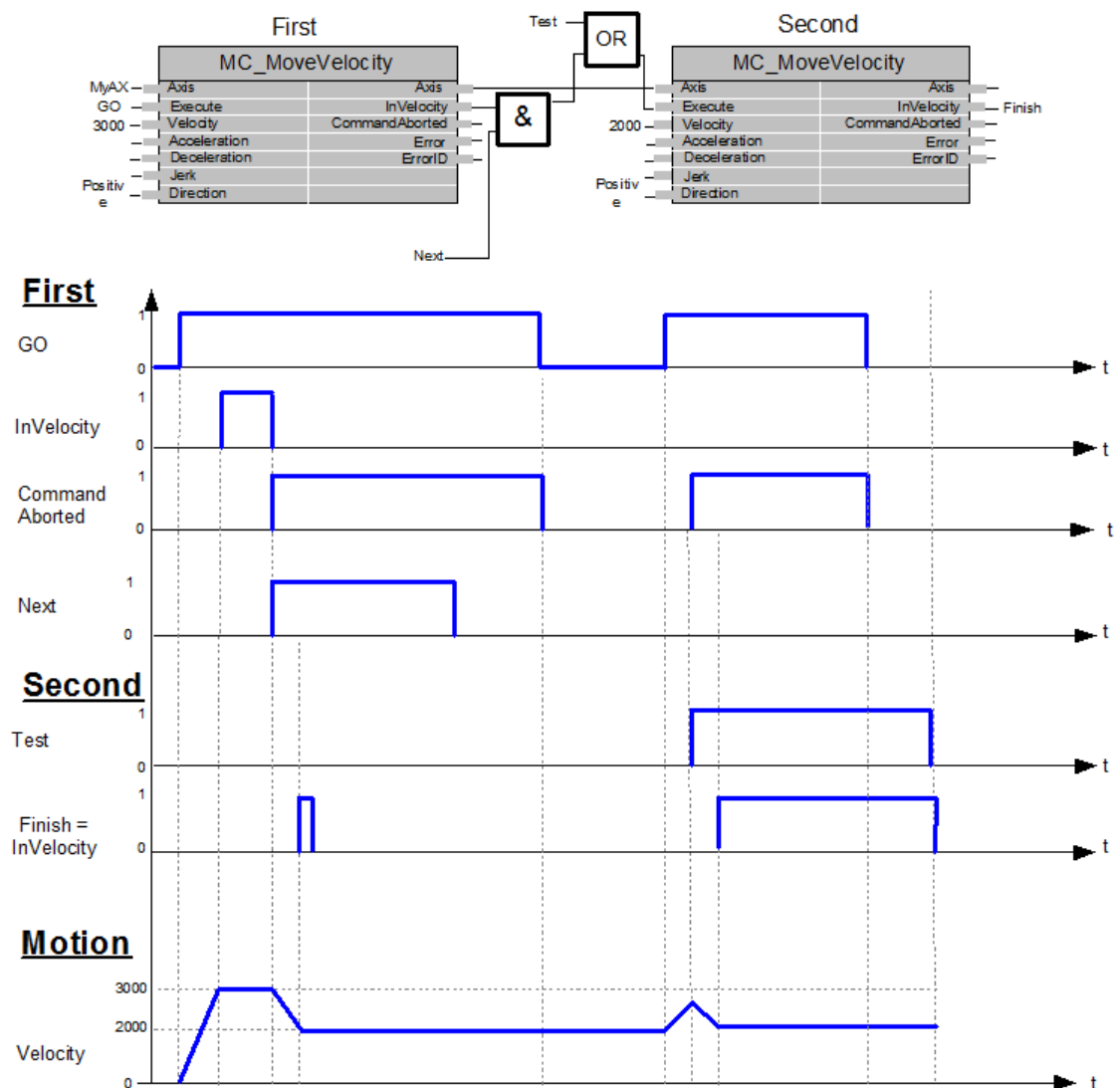


To stop the motion, the function block has to be interrupted by another function block issuing a new command.

- The signal “InVelocity” is set when the commanded velocity equals the velocity input.
- The signal “InVelocity” has to be reset when the block is aborted by another block or at the falling edge of “Execute”.
- In combination with MC_MoveSuperimposed, the output “InVelocity” stays TRUE once the velocity setpoint of the axis has reached the commanded velocity

Examples of the combination of two Function Blocks MC_MoveVelocity

- The left part of timing diagram illustrates the case if the second function block is called after the first one is completed. If first reaches the commanded velocity 3000 then the output First.InVelocity AND the signal Next causes the second function block to move to the velocity 2000.
- The right part of the timing diagram illustrates the case if the second function block starts the execution while the first function block is not yet InVelocity. The following sequence is shown: The first motion is started again by Go at the input First.Execute. While the first function block is still accelerating to reach the velocity 3000 the first function block will be interrupted and aborted because the test signal starts the Run of the second function block. Now the second function block runs and decelerates the velocity to 2000.



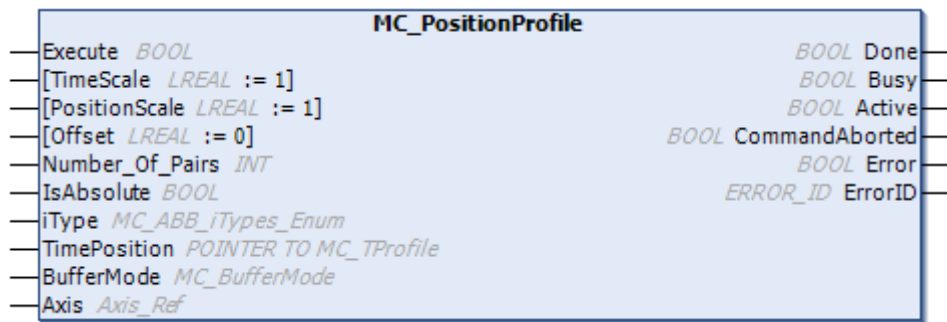
Input validation is done at the rising edge of Execute. If function block is in Active/Busy state, new value at input will not be validated. If value passed is invalid function block will continue execution with last valid value.

InOut:

Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
	Velocity	LREAL	[u/s] Value of the maximum velocity (not necessarily reached). Range: ≥ 0
	Acceleration	LREAL	[u/s ²] Value of the acceleration (increasing energy of the motor). Range: > 0 . If value = 0, Acceleration will be equal to parameter paraMaxAccelerationAppl
	Deceleration	LREAL	[u/s ²] Value of the deceleration (decreasing energy of the motor). Range: > 0 . If value = 0, Deceleration will be equal to parameter paraMaxDecelerationAppl
	Jerk	LREAL	[u/s ³] Value of the jerk. Range: ≥ 0
	Direction	MC_Direction	Positive, Negative, Current
	BufferMode	MC_Buffer-Mode	Not supported, default mcABORTING used
Out-put	InVelocity	BOOL	Commanded velocity is reached
	Busy	BOOL	The function block is not finished
	Active	BOOL	Indicates that the function block has control on the axis
	CommandAborted	BOOL	Command is aborted by another command from other PLCopen function block
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
Inout	Axis	Axis_Ref	Reference to axis

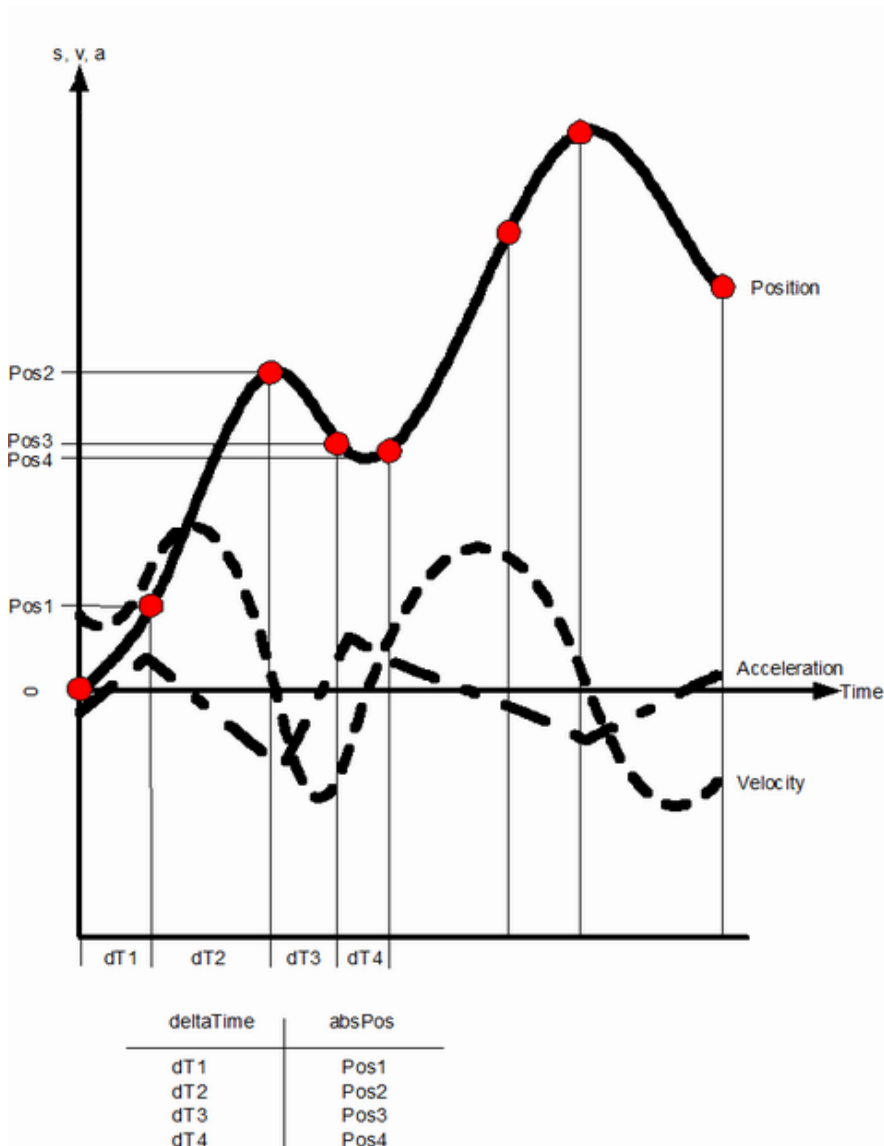
10.2.1.1.5.11 MC_PositionProfile (FB)



This function block commands a time-position locked motion profile.



- MC_TProfile is an ABB specific data type.
- This functionality does not mean it runs one profile over and over again: It can shift between different profiles.
- Alternatively to this function block, the CAM function block coupled to a virtual master can be used.

Example of Time/Position Profile



	The Time / Velocity and Time / Acceleration Profiles are similar to the Position Profile, with sampling points on the Velocity or Acceleration lines.
	Number_Of_Pairs input should always have equal to or less than actual pairs defined in TimePosition input

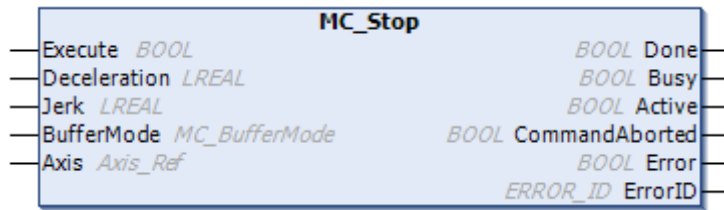
InOut:

Scope	Name	Type	Initial	Comment
Input	Execute	BOOL		Starts the function block at rising edge
	Time-Scale	LREAL	1	Overall time scaling factor of the profile. Range: >0
	Position-Scale	LREAL	1	Overall Position scaling factor. Range: PositionScale <> 0
	Offset	LREAL	0	Overall offset for profile, the profile result will be increased by Offset
	Number_Of_Pairs	INT		Number of sampling points, elements in TimePosition array. Range: >=2
	IsAbsolute	BOOL		Use absolute position values from profile. TRUE = Profile holds absolute position values
	iType	MC_ABB_iTypes_Enum		Type of interpolation. Possible values are: MCA_SPLINE_COMPLETE MCA_SPLINE_NATURAL MCA_POLY5 MCA_POLY3 MCA_LINEAR
	TimePosition	POINTER TO MC_TProfile		Reference to Time/Position. MC_TProfile is an ABB specific data type IsAbsolute = TRUE, interpolation_point = Actual position IsAbsolute = FALSE, interpolation_point = 0
	Buffer-Mode	MC_Buffer-Mode		Not supported, default mcABORTING used

Scope	Name	Type	Initial	Comment
Output	Done	BOOL		Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BOOL		The function block is not finished
	Active	BOOL		Indicates that the function block has control on the axis
	CommandAborted	BOOL		Command is aborted by another command from other PLCopen function block
	Error	BOOL		Signals that error has occurred within function block
	ErrorID	ERROR_ID		Error identification. For error details refer to Enumeration ERROR_ID
Inout	Axis	Axis_Ref		Reference to axis

10.2.1.1.5.12 MC_Stop (FB)

This function block commands a controlled motion stop and transfers the axis to the state “Stopping”.



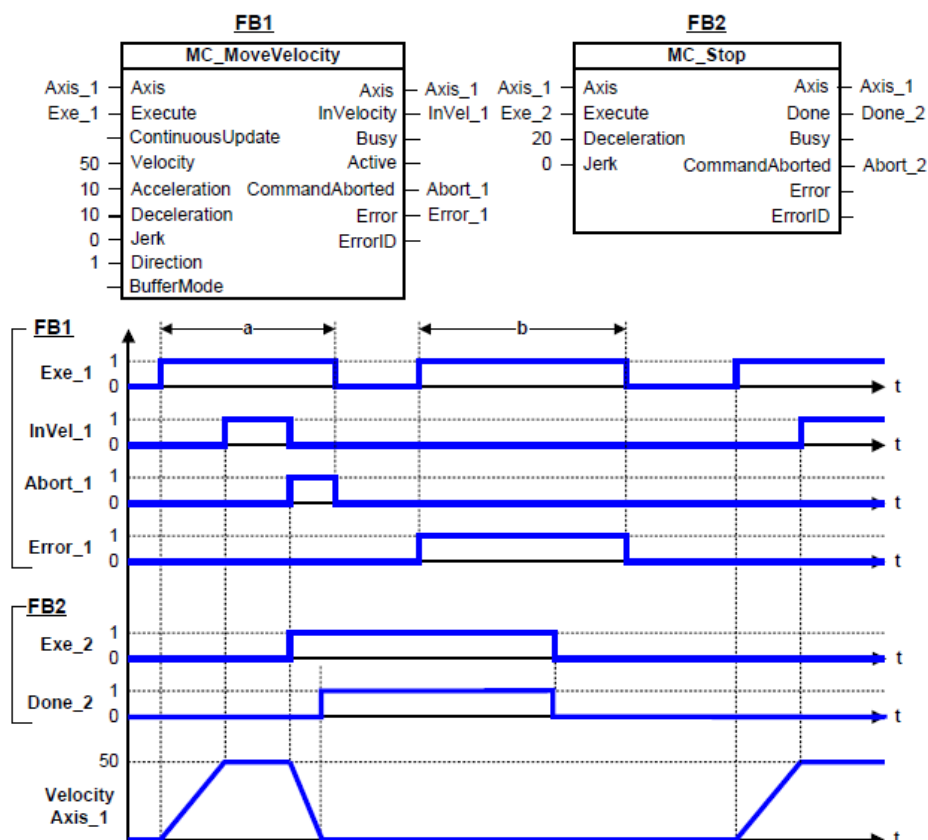
MC_Stop can not be aborted by another command. It can be aborted only by another MC_Stop function block

It aborts any ongoing function block execution. While the axis is in state Stopping, no other function block can perform any motion on the same axis. After the axis has reached velocity zero, the Done output is set to TRUE immediately. The axis remains in the state “Stopping” as long as Execute is still TRUE or velocity zero is not yet reached. As soon as “Done” is TRUE and “Execute” is FALSE the axis goes to state “STANDSTILL”.

Example

Behavior of MC_Stop in combination with MC_MoveVelocity

1. A rotating axis is ramped down with function block MC_Stop.
2. The axis rejects motion commands as long as MC_Stop parameter Execute = TRUE. Function block MC_MoveVelocity reports an error indicating the busy MC_Stop command.

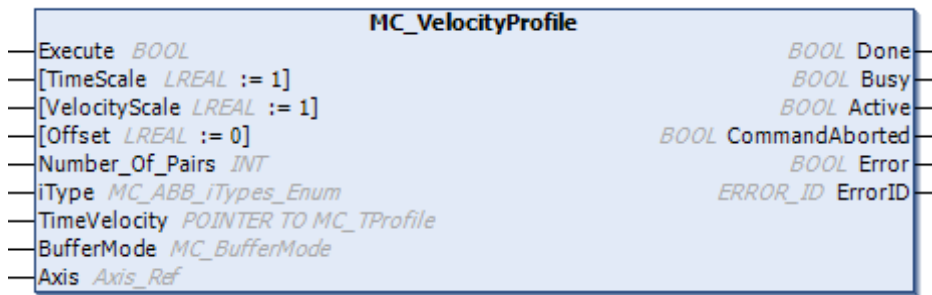


InOut:

Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
	Deceleration	LREAL	[u/s°2] Value of the deceleration (decreasing energy of the motor). Range: >0. If value = 0, Deceleration will be equal to parameter paraMaxDecelerationAppl
	Jerk	LREAL	[u/s°3] Value of the jerk. Range: >=0
	BufferMode	MC_Buffer-Mode	Not supported, default mcABORTING used
Out-put	Done	BOOL	Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BOOL	The function block is not finished
	Active	BOOL	Indicates that the function block has control on the axis
	CommandAborted	BOOL	Command is aborted by another command from other PLCopen function block
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	ERROR_ID	Error identification. For error details refer to Enumeration ERROR_ID
Inout	Axis	Axis_Ref	Reference to axis

10.2.1.1.5.13 MC_VelocityProfile (FB)

This function block commands a time-velocity locked motion profile.



- MC_TProfile is an ABB specific data type.
- This functionality does not mean it runs one profile over and over again: It can shift between different profiles.
- Alternatively to this function block, the CAM function block coupled to a virtual master can be used



When Done = TRUE (profile is completed), Axis will run with the last Velocity value.

InOut:

Scope	Name	Type	Initial	Comment
Input	Execute	BOOL		Starts the function block at rising edge
	TimeScale	LREAL	1	Overall time scaling factor of the profile. Range: >0
	VelocityScale	LREAL	1	Overall velocity scaling factor of the profile. Range: VelocityScale <> 0
	Offset	LREAL	0	Overall offset for profile, the profile result will be increased by Offset
	Number_Of_Pairs	INT		Number of sampling points, elements in Time-Velocity array. Range: >=2
	iType	MC_ABB_iTypes_Enum		Type of interpolation. Possible values are: MCA_SPLINE_COMPLETE MCA_SPLINE_NATURAL

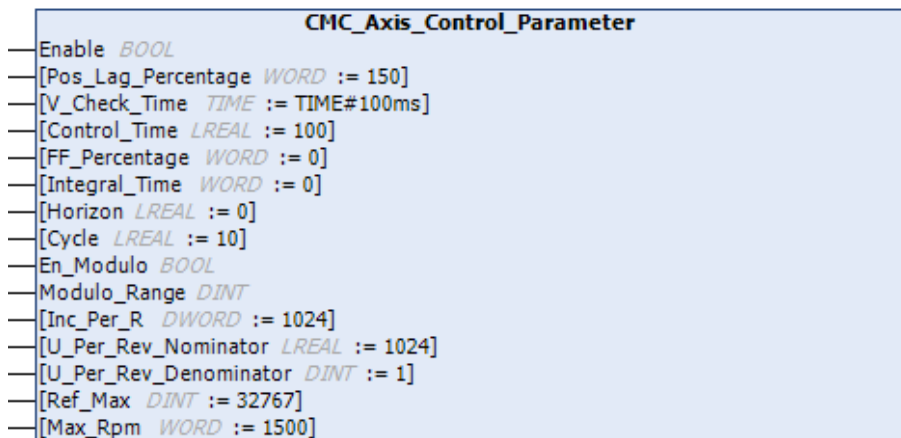
Scope	Name	Type	Initial	Comment
				MCA_POLY5 MCA_POLY3 MCA_LINEAR
	TimeVelocity	POINTER TO MC_TProfile		Reference to time/velocity. MC_TProfile is an ABB specific data type
	BufferMode	MC_BufferMode		Not supported, default mcABORTING used
Output	Done	BOOL		Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BOOL		The function block is not finished
	Active	BOOL		Indicates that the function block has control on the axis
	CommandAborted	BOOL		Command is aborted by another command from other PLCopen function block
	Error	BOOL		Signals that error has occurred within function block
	ErrorID	ERROR_ID		Error identification. For error details refer to Enumeration ERROR_ID
Inout	Axis	Axis_Ref		Reference to axis

10.2.1.2 CMC_Blocks

Central Motion control (CMC) related function and function blocks

10.2.1.2.1 CMC_Axis_Control_Parameter (FB)

Parameter block to provide basic information regarding the underlying axis behavior and to configure the closed loop control for CMC_Basic_Kernel.



Enable the function block before any PLCopen-block is used and this block has to be used in combination with CMC_Basic_Kernel.

Some explanations for the following parameters

To change parameters follow these steps:

1. MC_Power.Enable = FALSE, this will disable the axis.
2. CMC_Axis_Control_Parameter.Enable = FALSE.
3. Modify the parameter.
4. CMC_Axis_Control_Parameter.Enable = TRUE.

The control loop consists of a proportional factor and a predictive velocity feedforward.

The proportional factor is adjusted with "Control_Time".

On maximum speed and Feed-Forward = 0, the drive would move a distance which matches it's actual following-error in Control_Time ms.

The feed-forward is adjusted with a percentage-value which means: FF_Percentage = 100% => the following error will be 0. Recommendation is to use FF_Percentage <= 80, as higher values might result in overshoot.

The prediction horizon makes the feed-forward at high values more stable as it anticipates the drives delay.

When the drives step-response (to a velocity step) is measured and aligned to correspond a dead-time and a PT1,

- the Horizon will be exactly the dead-time
- the Control_Time corresponds to the time-constant of the PT1



The motor has to reach Max_Rpm when Ref_Max is issued as Speed_Reference, this relation is used to scale the control loop and following error supervision.



This function block passes all values into respective data types at the rising edge of input Enable. Henceforth, there are no outputs or validation of input values.

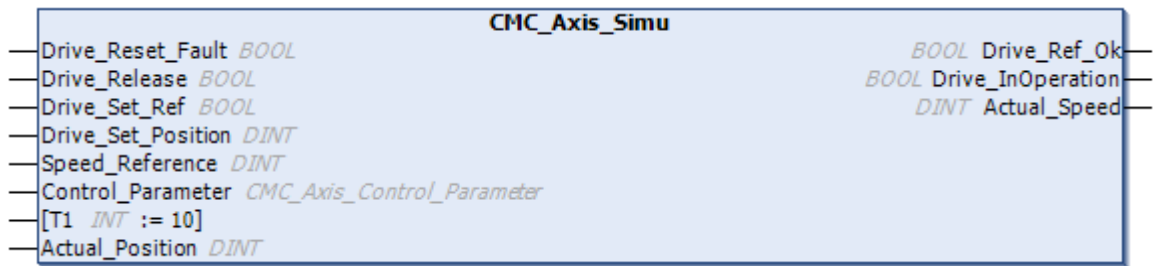
InOut:

Scope	Name	Type	Initial	Comment
Input	Enable	BOOL		New values are used on rising edge
	Pos_Lag_Percentage	WORD	150	Allowed percentage for following error, 100% will be reached with constant velocity at Max_Rpm with FF_Percentage = 0. Zero will switch off the monitoring
	V_Check_Time	TIME	TIME#100 ms	[ms] Tolerance time during which the velocity might deviate for 50%, zero will switch off the monitoring
	Control_Time	LREAL	100	[ms] Determines the gain for position control loop. A lower time means a larger proportional gain for position control loop. The value means: with FF_Percentage = 0, the drive will run Control_Time ms behind its position reference.(during movement with constant velocity) $20 \leq \text{Control_Time} \leq 100$ recommended
	FF_Percentage	WORD	0	Percentage for feed forward part of the control loop, <80% recommended. For larger values, the parameter Horizon needs to be used as the position will overshoot otherwise.
	Integral_Time	WORD	0	(ms), Integration time for position control loop, zero means no integral part is used.
	Horizon	LREAL	0	(ms) Gives a time in advance for the feed-forward. This could compensate reaction times. Horizon > 0 requires additional computing power.
	Cycle	LREAL	10	(ms) Cycle time of the PLC program.
	En_Modulo	BOOL		FALSE: The axis is a linear axis, TRUE: The positions will be calculated as modulo positions, the position value issued to the drive is still a 32bit value

Scope	Name	Type	Initial	Comment
	Mo- dulo_Ra nge	DINT		Distance for rollover, maximum value is 0x3FFFFFFF.
	Inc_Per_ R	DWORD	1024	Position resolution, Increments per revolution for actual position and Reference position
	U_Per_R ev_No- minator	LREAL	1024	Units per revolution = U_Per_Rev_Nominator/U_Per_Rev_Denominator
	U_Per_R ev_De- nomina- tor	DINT	1	Units per revolution = U_Per_Rev_Nominator/U_Per_Rev_Denominator
	Ref_Max x	DINT	32767	Maximum value for speed reference, has to be the value when Max_Rpm is reached
	Max_Rp m	WORD	1500	Maximum rotation per minute, has to be reached with Speed_Reference = Ref_Max

10.2.1.2.2 CMC_Axis_Simu (FB)

Simulation block can be used with CMC_Basic_Kernel to create a virtual axis.



The virtual axis has realistic behavior with a delayed response to velocity changes which result in a following error. Inputs and outputs have to be connected to the respective inputs and outputs of CMC_Basic_Kernel.

How to use the Axis Simulation

It is possible to use a simulated axis instead of a real drive. The axis simulation can be used in the following use cases:

- When the real drive is not available the simulation can be used to test all available motion functionalities to verify the application program.
- The simulation can be used to create a virtual master axis and synchronize other axes to it.

Homing will be possible if the limit-switches (data type CMC_Axis_IO) are simulated also. This is not done by CMC_Axis_Simu but could be realized in the PLC program

The drive velocity is simulated by PT1-Characteristic. The input T1 gives the time constant for this PT1 as multiple of the cycle time. All other properties are simulated according to the CMC_Axis_Control_Parameter



The value of the time behaviour from the axis simulation function block set by the input T1 has to be at least four times smaller than the value of the axis parameter CONTROL_TIME from the CMC_Axis_Control_Parameter function block

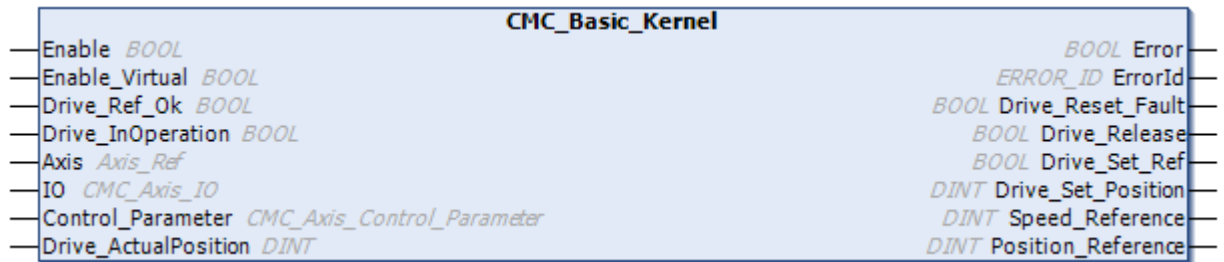
InOut:

Scope	Name	Type	Initial	Comment
Input	Drive_Reset_Fault	BOOL		Binary signal to be used for resetting the drive error, if applicable
	Drive_Release	BOOL		Activate the drive
	Drive_Set_Ref	BOOL		Activate homing
	Drive_Set_Position	DINT		Position to be used at homing

Scope	Name	Type	Initial	Comment
	Speed_Reference	DINT		Reference value for the drive
	Control_Parameter	CMC_Axis_Control_Parameter		Identical parameter which are used at CMC_Basic_Kernel, so the required answer to Speed_Reference is created
	T1	INT	10	Delay time in cycles
Inout	Actual_Position	DINT		Actual position [increments]
Output	Drive_Ref_Ok	BOOL		Indication for homing
	Drive_InOperation	BOOL		Indication that drive is running. The drive is switched on and is active
	Actual_Speed	DINT		Actual Position in incremenets

10.2.1.2.3 CMC_Basic_Kernel (FB)

The kernel function block is the fundamental part of the Central Motion Control axis implementation named Compact Motion. It performs floating point arithmetic for all calculations



For a central motion axis implementation the use of the Function Blocks CMC_Basic_Kernel and CMC_Axis_Control_Parameter

For more details of the Central motion control refer to the architecture chapter in the system technology document.



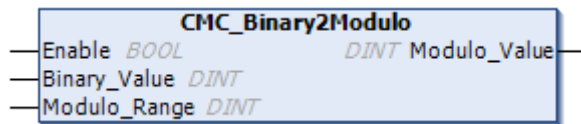
If the Function Block is stored in a RETAIN memory area, the connected AXIS has to be RETAIN

InOut:

Scope	Name	Type	Comment
Input	Enable	BOOL	Release of function block. Enable has to be set before new control parameters are released by CMC_Axis_Control_Parameter.
	Drive_Ref_Ok	BOOL	Indication for homing
	Drive_InOperation	BOOL	Indication that drive is running. The drive is switched on and is active
	Drive_ActualPosition	DINT	Actual Position in increments
Inout	Axis	Axis_Ref	Reference to the axis to be controlled
	Control_Parameter	CMC_Axis_Control_Parameter	Parameters for configuration and adjustment of the control loop
	IO	CMC_Axis_IO	By the structure IO (CMC_Axis_IO), some binary inputs are provided. The PLC program has to define a variable of type CMC_Axis_IO and to assign the inputs.
Output	Error	BOOL	Signals that an error has occurred within the function block.
	ErrorId	ERROR_ID	The error codes ErrorId also sets the output-bit Error=TRUE and sets the axis in state ERROR_STOP. To allow a new movement

Scope	Name	Type	Comment
			<p>the error-codes ErrorId require that either the axis is disabled/enabled by MC_Power or the error reset is disabled/enabled by MC_Reset.</p> <p>The error codes ErrorID_WARNING will not set Error = TRUE, and will not set the axis to ERROR_STOP. The error codes ErrorID_WARNING do not require the MC_Reset or MC_Power. It is possible the axis is stopped and ongoing motion is aborted by a WARNING. The value will be shown until: + An other error or warning occurs + MC_Reset or MC_Power is used</p>
	Drive_Reset_Fault	BOOL	Binary signal to be used for resetting the drive error, if applicable
	Drive_Release	BOOL	Activate the drive
	Drive_Set_Ref	BOOL	Activate homing
	Drive_Set_Position	DINT	Position to be used at homing
	Speed_Reference	DINT	Reference value for the drive
	Position_Reference	DINT	Position reference for the drive in increments
Input	Enable_Virtual	BOOL	Use the axis as virtual axis. Block inputs which are usually received from the real axis are ignored. Required values are generated internally

10.2.1.2.4 CMC_Binary2Modulo (FB)



Convert a 32 bit value (Position_Reference) to Modulo_Range.

Purpose: Convert Value to $0 \leq \text{Modulo_Value} \leq \text{Modulo_Range}$

Precondition: Enable the block together with CMC_Modulo2Binary, use the two blocks as a pair

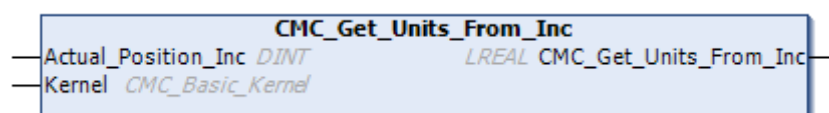
Use case: A drive configured as a rollover axis expects the position reference in a modulo style. Use this block to convert the position reference.

InOut:

Scope	Name	Type	Comment
Input	Enable	BOOL	Enable the block execution
	Binary_Value	DINT	32 bit value
	Modulo_Range	DINT	Max used value for Modulo_Value
Output	Modulo_Value	DINT	Modulo value

10.2.1.2.5 CMC_Get_Units_From_Inc (FUN)

This function converts the drive's position value (DINT) which is exchanged between drive and PLC to the corresponding scaled position unit (LREAL) which is used by the PLCopen function blocks.

**Use case**

The drive or an IO-device is used to capture an axis encoder position in relation to a binary signal (touch trigger). This position is delivered in [increments]. If then the position is to be used in the PLCopen context, a [unit] position is required. It can be difficult to calculate this unit-position. Not just the scaling for position units has to be considered but also the position might have experienced several correction measures. Measures like "SetPositionContinuous" or corrections due to modulo position overrun. To create the unit-value which matches a certain increment value, the function CMC_Get_Units_From_Inc has to be used.

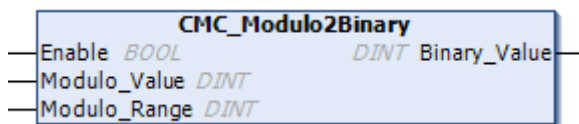
Return value

The position [u], which describes exactly the same position as Actual_Position_Inc, just transferred in to the axis coordinate system and delivered in [u].

InOut:

Scope	Name	Type	Comment
Return	CMC_Get_Units_From_Inc	LREAL	
Input	Actual_Position_Inc	DINT	A position [increments], for example captured by the drive as result for a touch trigger
Inout	Kernel	CMC_Basic_Kernel	Kernel block instance which belongs to the specific axis.

10.2.1.2.6 CMC_Modulo2Binary (FB)



Convert a <32 bit value to 32 bit for use as Actual_Position.

Purpose: Convert Value which is $0 \leq \text{Value} \leq \text{Modulo_Range}$ to Binary_Value (with 32-Bit overflow)

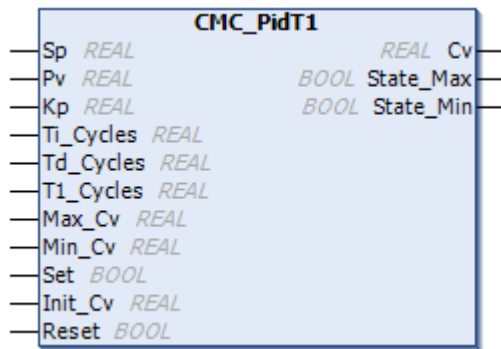
Precondition: Enable the block together with CMC_Binary2Modulo, use the two blocks as a pair

Use case: A drive configured as a rollover axis delivers the actual position in a modulo style while the CMC_Basic_Kernel expects a binary, or an absolute encoder with <32 bit position value is used as actual position. Use this block to convert the position to a 32 bit value.

InOut:

Scope	Name	Type	Comment
Input	Enable	BOOL	Enable the block execution. Enable = FALSE => Binary_Value := Modulo_Value
	Modulo_Value	DINT	Actual value
	Modulo_Range	DINT	(Max allowed value for Modulo_Value)+1
Output	Binary_Value	DINT	Binary value

10.2.1.2.7 CMC_PidT1 (FB)



PIDT1 Controller

- The PI controller changes its output Cv (manipulated variable) until input Pv (controlled variable) is equal to input Sp (command variable).
- Additional pre filter for DT1 - Component available (optional).
- I or DT1 components can be switched off by setting the respective time equal to zero.

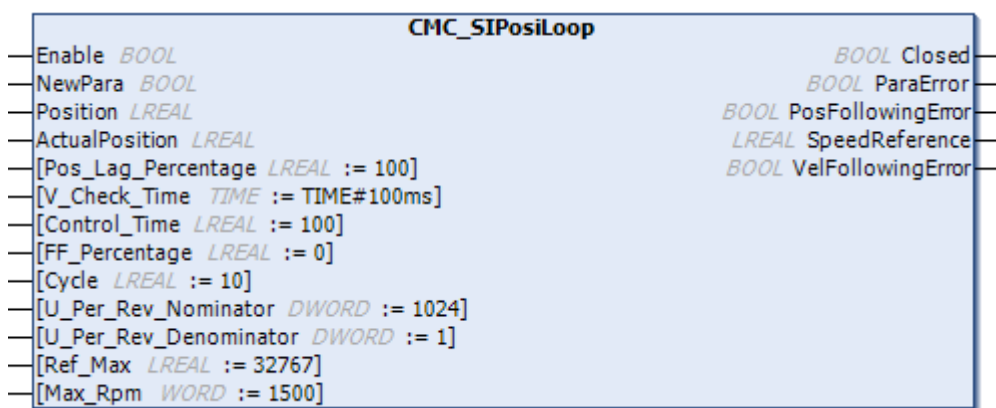
InOut:

Scope	Name	Type	Comment
Input	Sp	REAL	Command variable (set point)
	Pv	REAL	ProcessValue, Actual Value
	Kp	REAL	Proportional coefficient, specified as a percentage value
	Ti_Cycles	REAL	Integral action time scaled to the PLC cycle time
	Td_Cycles	REAL	Derivative action time scaled to the PLC cycle time
	T1_Cycles	REAL	Returning time scaled to the PLC cycle time
	Max_Cv	REAL	High limit for the manipulated variable Cv
	Min_Cv	REAL	Low limit for the manipulated variable Cv
	Set	BOOL	Enable for setting to initial value Init
	Init_Cv	REAL	Initial value for the manipulated variable Cv
	Reset	BOOL	Reset of the manipulated variable Cv to 0

Scope	Name	Type	Comment
Out-put	Cv	REAL	Output for the manipulated variable Control Value
	State_Max	BOOL	High limit has been reached
	State_Min	BOOL	Low limit has been reached

10.2.1.2.8 CMC_SIPosiLoop (FB)

The function block can be used for a simple interpolation. Alternatively, the function block CMC_SInterpolation can be used. A “stand alone” position control loop



This function block has to be used along with function block CMC_SInterPolation



This block has to be called within the REAL-TIME task

Behavior of inputs

InSync	Active	Behavior
FALSE	x	Output Position = ActualPosition
TRUE	FALSE	Interpolates output Position to reach TargetPosition with the given velocity and acceleration.
TRUE	TRUE	Ramps down to velocity = 0.

Behavior of outputs

InSync	Active	Behavior
FALSE	TRUE	Function block is activated, position and velocity is not yet reached.
TRUE	FALSE	Function block is activated, position is reached, output Position = TargetPosition.
FALSE	TRUE	Function block is either disabled or stopped, velocity

Different use cases for this block:**Use in Combination with MC_MoveByExternalReference:**

A positioning axis can be created with modifying the parameters for positioning “on the fly”, without the need of a certain statemachine to follow.

Use in Combination with MCA_SetDynamicFollower:

The function block will smooth the movement when a group has to follow a conveyor.

It will create defined ramps to accelerate to the conveyors position and also can be used to prevent a position jump when switching between 2 conveyors or before switching of the follower.

Use to Create a Simple, basic positioning Axis:

A very basic, simple positioning axis can be created by using this block without the PLCopen blocks.

Be aware that there is not additional check if the axis really follows, also no scaling for the position is included.

If function block CMC_SIPosiLoop is used, it will check for position or velocity following error.

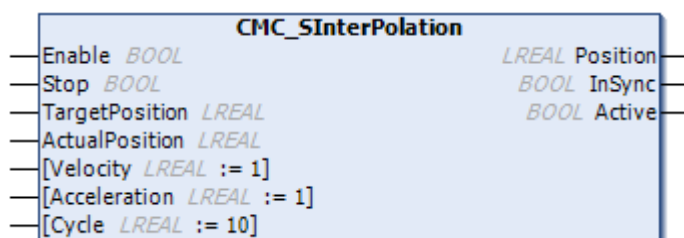
InOut:

Scope	Name	Type	Initial	Comment
Input	Enable	BOOL		Level triggered, close = 1, open = 0, a rising edge required to clear any error condition
	NewPara	BOOL		At least one rising edge required to set parameter, TRUE: internal values calculated from input parameters, level triggered
	Position	LREAL		[u] Reference position
	ActualPosition	LREAL		[u] Actual position
	Pos_Lag_Percentage	LREAL	100	[%] Allowed percentage for following error, 100% will be reached with constant velocity at Max_Rpm with FF_Percentage = 0. Zero will switch off the monitoring
	V_Check_Time	TIME	TIME#100ms	[ms] Tolerance time during which the velocity might deviate for 50%, zero will switch off the monitoring

Scope	Name	Type	Initial	Comment
	Control_Time	LREAL	100	[ms] Determines the gain for position control loop
	FF_Percentage	LREAL	0	[%] Percentage for feed forward part of the control loop
	Cycle	LREAL	10	[ms] Cycle time of the PLC program
	U_Per_Rev_Nominator	DWORD	1024	Units per motor revolution = U_Per_Rev_Nominator/U_Per_Rev_Denominator
	U_Per_Rev_Denominator	DWORD	1	Units per motor revolution = U_Per_Rev_Nominator/U_Per_Rev_Denominator
	Ref_Max	LREAL	32767	Maximum value for SpeedReference, has to be the value when Max_Rpm is reached
	Max_Rpm	WORD	1500	Maximum rotation per minute, has to be reached with SpeedReference = Ref_Max
Output	Closed	BOOL		Position control loop activated and closed
	ParaError	BOOL		Wrong parameter
	PosFollowingError	BOOL		Following error detected
	SpeedReference	LREAL		Speed reference. Range -Ref_Max <= SpeedReference <= Ref_Max
	VelFollowingError	BOOL		Velocity error detected

10.2.1.2.9 CMC_SInterPolation (FB)

The function block can be used for a simple point-to-point interpolation.



It can be combined with the function block CMC_SIPosiLoop as position control loop.

The function block allows to create a very simple, basic axis for linear movement. The function block can be either used independent to create a basic axis with/without position control loop or it can be used in combination with PLCopen function blocks.

The function block creates a positioning interpolation towards TargetPosition and uses the given velocity and acceleration values. The function block has to be used within the real-time cycle. The result is given to output Position. The TargetPosition and also Velocity and Acceleration can be changed anytime and will be used at once. With Enable = FALSE, the function block sets the output Position to ActualPosition, this is similar to an open loop.

Behavior of inputs:

Enable	Stop	Behavior
FALSE	x	Output Position = ActualPosition
TRUE	FALSE	Interpolates output Position to reach TargetPosition with the given velocity and acceleration.
TRUE	TRUE	Ramps down to velocity = 0.

Behavior of outputs:

InSync	Active	Behavior
FALSE	TRUE	Function block is activated, position and velocity is not yet reached.
TRUE	FALSE	Function block is activated, position is reached, output Position = TargetPosition.
FALSE	TRUE	Function block is either disabled or stopped, velocity

Different use cases for this block:

Use in Combination with MC_MoveByExternalReference:

A positioning axis can be created with modifying the parameters for positioning “on the fly”, without the need of a certain statemachine to follow.

Use to Create a Simple, basic positioning Axis:

A very basic, simple positioning axis can be created by using this block without the PLCopen blocks.

Be aware that there is not additional check if the axis really follows, also no scaling for the position is included.

If function block CMC_SIPosiLoop is used, it will check for position or velocity following error.



This block has to be called within the REAL-TIME task

InOut:

Scope	Name	Type	Initial	Comment
Input	Enable	BOOL		Level triggered, Enables the interpolation
	Stop	BOOL		Stop, level triggered, interpolate down to velocity = 0
	TargetPosition	LREAL		[u] Target position
	ActualPosition	LREAL		[u] Actual position
	Velocity	LREAL	1	[u/s] Value of the maximum velocity (not necessarily reached). Range: >0
	Acceleration	LREAL	1	[u/s^2] Value of the acceleration and deceleration. Range: >0
	Cycle	LREAL	10	[ms] Cycle time of the PLC program.
Output	Position	LREAL		[u] New absolute position
	InSync	BOOL		Position reached TargetPosition
	Active	BOOL		Interpolation active

10.2.1.3 Data types

10.2.1.3.1 Enums

10.2.1.3.1.1 ERROR_ID (ENUM)

This enumeration contains all the errors generated for the motion control function blocks.

InOut:

Name	Initial	Comment
MC_Ok	0	No error
Wrong_State	1	PLCOpen block was activated, but axis not in a allowed state to start, or axis left the moving state without a PLCOpen block requiring the state change
Drive_Problem	2	Drive indicates an error. e.g. Tripped
Parameter_Exceeds_Limit	3	Input parameter at PLCopen block exceeds allowed limits
No_Field_Access	4	No fieldbus connection to drive
Bus_Problem	5	Not used
Abs_Switch_Error	6	During homing, limit switch not according to moving direction. e.g. the positive switch occurred when moving in negative direction
Timeout	7	Timeout in block execution
NAK	8	Parameter access not applicable
MC_TimeLimitExceeded	9	Used by function blocks with TimeLimit
MC_DistanceLimitExceeded	10	Used by function blocks with DistanceLimit
MC_TorqueLimitExceeded	11	Used by function blocks with TorqueLimit
Not_Implemented	12	Functionality not implemented for certain axis type
ErrorID_POSITION_FOLLOW	101	Following error, caused by > position error => ERRORSTOP.(parameter POS_LAG_PERCENTAGE)
ErrorID_POSSW	102	Positive software limit switch => ERRORSTOP. The actual position did exceed the positive Software limit switch position. This supervision has to be activated with MC_WriteParameter.

Name	Initial	Comment
ErrorID_NEGSW	103	Negative software limit switch => ERRORSTOP. The actual position did exceed the negative Software limit switch position. This supervision has to be activated with MC_WriteParameter.
ErrorID_VELOCITY_FAULT	104	The measured velocity and commanded velocity are > 50% (related to maximum velocity) apart, for a certain time =>ERRORSTOP (parameter V_CHECK-TIME)
ErrorID_INTERPOLATION_FAULT	105	following error, caused by interpolation problem =>ERRORSTOP. Position following error occurred, but reason most likely a interpolation problem, not drive problem (e.g. CAM Table, position step).
ErrorID_WARNING_VELOCITYLIMIT	110	Velocity or acceleration/deceleration are in limitation, set by parameter EnableLimitVelocity, MaxVelocityAppl, MaxDecelerationAppl
ErrorID_WARNING_POSITIONLIMITPOS	111	Position is in limitation towards position limit (SWLimit2DecPos), axis decelerates near positive software limit switch
ErrorID_WARNING_POSITIONLIMITNEG	112	Position is in limitation towards position limit (SWLimit2DecNeg)., axis decelerates near negative software limit switch
ErrorID_WARNING_POSITIONOVERRUN	113	A linear axis created a 32bit position overrun (> 2147483647 u=>inc) =>configure modulo
ErrorID_WARNING_ABORT	114	Axis aborted due to too large position gap due to velocity limitation
ErrorID_WARNING_MOVEMENT_DIRECTION	115	Either positive or negative direction blocked by MC_Power

10.2.1.3.1.2 MC_ABB_iTypes_Enum (ENUM)

Different types of interpolation to be used for cam table or time based profile movement.

InOut:

Name	Comment
MCA_SPLINE_COMPLETE	Cubic spline interpolation without jerk, start and end of profile with velocity = 0.
MCA_SPLINE_NATURAL	Cubic spline interpolation without jerk
MCA_POLY5	Polynomial interpolation with linear acceleration between interpolation points.
MCA_POLY3	Polynomial interpolation with linear velocity between interpolation points.
MCA_LINEAR	Linear interpolation with constant velocity between interpolation points.

10.2.1.3.1.3 MC_BufferMode (ENUM)

Only mcABORTING supported

InOut:

Name
mcABORTING
mcBUFFERED
mcBLENDINGlow
mcBLENDINGprevious
mcBLENDINGnext
mcBLENDINGhigh

10.2.1.3.1.4 MC_Direction (ENUM)

Enumeration to determine the moving direction to be used for some PLCopen blocks. Some of the values are just valid for a modulo axis, as the direction for a positioning movement for a linear axis is determined by the position.

InOut:

Name	Comment
DEFAULT	
POSITIVE	Default direction
SHORTEST	Positive direction
NEGATIVE	Shortest direction
CURRENT	Negative direction
POSITIVE_STOP	Current direction
NEGATIVE_STOP	Move and stop positive, just valid with a modulo axis and if the actual movement is already positive. An additional modulo distance will be moved if it is not possible to ramp down to the given position.
CURRENT_STOP	Move and stop negative, just valid with a modulo axis and if the actual movement is already negative. An additional modulo distance will be moved if it is not possible to ramp down to the given position. Move and stop in the current direction, just valid with a modulo axis. An additional modulo distance will be moved if it is not possible to ramp down to the given position.

10.2.1.3.1.5 MC_Homing_Direction (ENUM)

Homing Directions

InOut:

Name	Comment
MC_SwitchNegative	Switch Negative
MC_SwitchPositive	Switch Positive
MC_Positive	Positive
MC_Negative	Negative

10.2.1.3.1.6 MC_Homing_Edge (ENUM)**InOut:**

Name	Comment
MC_EdgeOn	Edge On
MC_EdgeOff	Edge Off
MC_On	On
MC_Off	Off

10.2.1.3.1.7 MC_Homing_Mode (ENUM)**InOut:**

Name	Comment
MC_REFPULSE	Reference pulse
MC_DIRECT	Direct

10.2.1.3.1.8 MC_Source (ENUM)

Determine if a slave axis would follow the masters actual position (mcActualValue) or reference position (mcSetValue). If the master is controlled by the PLC (as a real axis or virtual axis) it is preferable to use mcSetValue.

InOut:

Name	Comment
mcActualValue	Actual value
mcSetValue	Set value

10.2.1.3.2 Structs

10.2.1.3.2.1 Axis_Parameter (STRUCT)

PLCOpen parameter and supplier specific parameter. Limitations, position control, modification of behaviour

InOut:

Name	Type	Initial	Comment
posWindow	LREAL	10	The limit for the axis to reach the target position and indicate "Done" in a positioning move
v_Window	LREAL	10	The limit for the axis to reach the target velocity and indicate "InVelocity" in a velocity move
busdelay	TIME	TIME#1s0ms	Bus delay time
paraSWLimitPos	LREAL	2147483647	PLCOpen Parameter 02 Positive Softwarelimit switch position.
paraSWLimitNeg	LREAL	-2147483647	PLCOpen Parameter 03 Negative Softwarelimit switch position.
paraEnableLimit-Pos	BOOL		PLCOpen Parameter 04 Enable positive software limit switch.
paraEnableLimit-Neg	BOOL		PLCOpen Parameter 05 Enable negative software limit switch.
paraEnablePosLagMonitoring	BOOL	TRUE	PLCOpen Parameter 06 Enable monitoring of position lag (following error).
paraMaxPosition-Lag	LREAL		PLCOpen Parameter 07 Maximal position lag.
paraMaxVelocitySystem	LREAL	16#3FFFFFFF	PLCOpen Parameter 08 Maximal allowed velocity of the axis in the motion system
paraMaxVelocityAppl	LREAL	16#3FFFFFFF	PLCOpen Parameter 09 Maximal allowed velocity of the axis in the application.
paraMaxAccelerationSystem	LREAL	16#3FFFFFFF	PLCOpen Parameter 12 Maximal allowed acceleration of the axis in the motion system.
paraMaxAccelerationAppl	LREAL	16#3FFFFFFF	PLCOpen Parameter 13 Maximal allowed acceleration of the axis in the application.
paraMaxDecelerationSystem	LREAL	16#3FFFFFFF	PLCOpen Parameter 14 Maximal allowed deceleration of the axis
paraMaxDecelerationAppl	LREAL	16#3FFFFFFF	PLCOpen Parameter 15 Maximal allowed deceleration of the axis.
paraMaxJerk	LREAL	16#7FFFFFFF	PLCOpen Parameter 16 Maximal allowed jerk of the axis.
paraGearBox_Numerator	DINT	1	ABB specific parameter 2001. Used for Central Motion Control implementation: Gearbox modifier to MODULO_RANGE

Name	Type	Initial	Comment
paraGearBox_Denominator	DINT	1	ABB specific parameter 2002. Used for Central Motion
paraEnableLimit2Decelerate	BOOL		ABB specific parameter 2003. Enable software limit switches to decelerate
paraEnableLimitAbort	BOOL		ABB specific parameter 2004. Enable that software limit switches will abort ongoing movement FALSE = Limits position and velocity, decelerates and shows a warning until the position limit is reached, then ERROR STOP TRUE = Switches off any ongoing motion and decelerates to the position limit, then ERROR STOP
paraEnableLimitVelocity	BOOL		ABB specific parameter 2005. Enable that velocity is limited
paraSWLimit2DecPos	LREAL	2147483647	ABB specific parameter 2006 Positive Software limit position, used with EnableLimit2Decelerate.
paraSWLimit2DecNeg	LREAL	-2147483647	ABB specific parameter 2007 Negative Software limit position,, used with EnableLimit2Decelerate.
paraMaxPosition-Gap	LREAL		ABB specific parameter 2008. Maximum allowed position to be lost by paraEnableLimitVelocity, will stop the ongoing movement
paraFilterVariant	INT		Filter for actual velocity, 0 = PT1, 1 = LinearRegression
paraFilterTime	INT	10	Time in PLC cycles, used with paraFilterVariant
paraFilterForecast	INT	0	Time in PLC cycles, used with paraFilterVariant = 1
paraReverseDirection	INT	0	Changes the direction for actual and reference positions based on the mode selected. 0 = normal direction, 1 = reverse input position, 2 = reverse output position and speed reference, 3 = reverse both
paraEarlyClosed-Loop	BOOL	FALSE	TRUE: hold the position when DRIVE_RELEASE is set (not wait for DRIVE_INOPERATION=TRUE)
paraLateOpen-Loop	BOOL	FALSE	TRUE: hold the position until DRIVE_INOPERATION=FALSE
paraWarningDirectionAsError	BOOL	FALSE	TRUE: if MC_Power has disabled a direction, a movement in this direction set the axis to ERRORSTOP, FALSE: just interrupt the movement
position_control	CMC_Pos_Control		Values related to position control loop

10.2.1.3.2.2 Axis_Ref (STRUCT)

Holds the main information regarding the axis. It is used to identify an axis and connect the various PLCopen blocks and KERNEL block which belong to the specific axis.

InOut:

Name	Type	Comment
user	CMC_Axis_User	User Axis details
inout	CMC_Axis_InOut	This structure should handle the in-out values to/from the open-motion function blocks to the fieldbus or the internal motion software. It represents a neutral interface
actual	CMC_Axis_Actual	Some actual values like positions and velocity per cycle
parameter	Axis_Parameter	Parameter which are written/read with MC_WriteParameter/MC_ReadParameter blocks and also parameter used for position control loop
expert	Expert	Expert parameters available to use only with ' Motion Wizard ' project.

10.2.1.3.2.3 CMC_Axis_Actual (STRUCT)

Some actual axis values which are used inside position control loop

InOut:

Name	Type	Comment
POSITION	LREAL	Actual position [u]
CONTROL_POSITION	LREAL	Used control position [u], could be limited by software limit switch or velocity limitation
D_XS	LREAL	Actual reference velocity, [u/cycle]
DD_XS	LREAL	Actual acceleration/deceleration (without superimposed movement)
D_XI	LREAL	Actual velocity, [u/cycle]
D_XSS	LREAL	Actual following error, [u]

Name	Type	Comment
ACT_POS_GAP	LREAL	Actual position gap, caused by velocity limitation
REFERENCE_PROP	LREAL	Proportional part for speed reference, calculated from D_XSS
REFERENCE_FF	LREAL	Feed forward for speed reference, calculated from D_XS
REFERENCE_ITG	LREAL	Integral part for speed reference, calculated from D_XSS

10.2.1.3.2.4 CMC_Axis_IO (STRUCT)

The limit switches are interpreted in positive logic. The logic for the reference encoder could be chosen

InOut:

Name	Type	Comment
limitSwitchPos	BOOL	Limit switch positive
limitSwitchNeg	BOOL	Limit switch negative
absRefSwitch	BOOL	Reference switch

10.2.1.3.2.5 CMC_Axis_InOut (STRUCT)

This structure should handle the in-out values to/from the open-motion function blocks to the fieldbus or the internal motion software. It represents a neutral interface.

InOut:

Name	Type	Comment
basicImp	zCMC_AXIS_IPO_DATA	Parameter data for a basic positioning or velocity move
used_destination	LREAL	Needed for coordinated movement
tp_ref	POINTER TO zCMC_TPro-file_TABLE	Parameter data for position profile/velocity profile/acceleration profile
pp_ref	POINTER TO zCMC_PPro-file_TABLE	Parameter data for cam table
cp_ref	POINTER TO MC_Cam_Id	Additional parameter for cam table to modify the movement
profile_scale	zCMC_AXIS_PRO-FILE_SCALE	Profile Scale
gear_scale	zCMC_AXIS_GEAR_SCALE	Gearing Scale
start_mode	zCMC_AXIS_START_MODES	Start Modes
superImp	zCMC_AXIS_IPO_DATA	Parameter data for a superimposed move
override	zCMC_AXIS_OVERRIDE	Axis overrides
phasing	zCMC_AXIS_PHASING	Phasing may be buffered, but not followed by buffered => single
correction	zCMC_AXIS_IPO_DATA	Parameter data for a MCA_SetPositionContinuous
release	BOOL	Release
quit	BOOL	Quit
setPosition	BOOL	Set Position
setRef	BOOL	Set Reference
refresh_Position	BOOL	Refresh position
ignoreLimit	BOOL	Limit switches allowed during homing
actual_position	LREAL	Used to communicate the actual position to/from MC-function blocks, identical to actual.POSITION
actual_velocity	LREAL	Actual velocity
actual_control_position	LREAL	Used to communicate the control position to/from MC-function blocks
actual_D_XS	LREAL	Actual control velocity, related to CYCLE
MC_error	zCMC_ABB_ERRORINFO	Error Information
actPhaseShift	LREAL	Actual phase shift
actCorrection	LREAL	Actual correction
axis_error_code	WORD	Axis error codes
drive_error_code	WORD	Drive error codes
remote_ok	BOOL	Remote operation OK
start_level	BOOL	Start Level
inSync	BOOL	In synchronisation
in_window	BOOL	In Window

Name	Type	Comment
closed	BOOL	Closed
ref_ok	BOOL	Reference ok
in_velocity	BOOL	In Velocity
pos_profile_ready	BOOL	Position profile is ready
cam_profile_ready	BOOL	Cam profile is ready
start_mode_error	BOOL	Error in start mode
axis_error	BOOL	Axis error
drive_error	BOOL	Drive error
enablePositive	BOOL	Enable positive movement
enableNegative	BOOL	Enable negative movement

10.2.1.3.2.6 CMC_Axis_User (STRUCT)

User Axis details

InOut:

Name	Type	Comment
number	INT	Parameter number
typ	INT	1 = ACS800_dc, 2=ACSM1_dc, 3=ACS350_dc 11=EASY
name	STRING	Name

10.2.1.3.2.7 CMC_Pos_Control (STRUCT)

Some parameter which are used for position control. These Parameters are calculated from CMC_Axis_Control_Parameter, They can be modified directly by accesing this structure. Any modification would be effective immediately. A rising edge at "CMC_Axis_Control_Parameter" will overwrite the manual modifications. The sepearte values for backward movement will be identical to the forward values and can be modified manually if required.

InOut:

Name	Type	Initial	Comment
KP	LREAL	1	Proportional gain
KF	LREAL	1	Used feed forward gain
KP_BACK	LREAL	1	Proportional gain for backward movement
KF_BACK	LREAL	1	Used feed forward gain for backward movement
KF_100	LREAL	1	Feed forward gain of 100%
TI	INT		Integration time for PI-Control. Just P Control is used when TI = 0

Name	Type	Initial	Comment
KI	INT		Proportional gain for integral part of controller
MAX_TIME	TIME		Maximum time for velocity supervision
D_XS_MAX	LREAL		Maximum system velocity in u/cycle
REF_MAX_POS	DINT	32767	Maximum value to be used for SPEED_REFERENCE in positive direction
REF_MAX_NEG	DINT	32767	Maximum value to be used for SPEED_REFERENCE in negative direction

10.2.1.3.2.8 Expert (STRUCT)

Expert parameters available to use only with ' Motion Wizard ' project.

InOut:

Name	Type	Comment
Virtual	BOOL	Specifies if the axis is virtual or real
Modulo	BOOL	True if Axis is configured as Modulo Axis.
ModuloValue	LREAL	Modulo Value for Axis
EncoderResolution	DWORD	Increments Per Revolution
ScalingNumerator	LREAL	Scaling Numerator
ScalingDenominator	DINT	Scaling Denominator
Error	BOOL	Error Bit for axis
ErrCode	DINT	Error Code for Axis
ActPosition	LREAL	Actual Position of Axis
ActVelocity	LREAL	Actual Velocity of Axis
OperatingMode	BYTE	Operating Mode (Drive Operating Mode)
Kernel	iBasicKernel	Kernel Block
AxisIO	CMC_AXIS_IO	Axis IO

Name	Type	Comment
ECAT	iBasicEthercat	Ethercat Application Block
AxisStatus	zCMC_STATES	Axis State
SetPosition	LREAL	Set Position Reference (MC Source: Set Position)
TouchProbeFunction	UINT	TouchProbeFunction for Touch Probe Application
TouchProbeStatus	UINT	TouchProbe Status for Touch Probe Application
TouchProbePositionPos1	DINT	TouchProbe 1 Rising Edge
TouchProbePositionNeg1	DINT	TouchProbe 1 Falling Edge
TouchProbePositionPos2	DINT	TouchProbe 2 Rising Edge
TouchProbePositionNeg2	DINT	TouchProbe 2 Falling Edge
TargetTorque	INT	EtherCAT object 16#6071, Target torque value for the torque controller in profile torque mode
MaxProfileVelocity	UDINT	EtherCAT object 16#607F, Max profile velocity

10.2.1.3.2.9 MCA_Parameter_Struct (STRUCT)

Used from MCA_WriteParameterList and MCA_ReadParameterList

InOut:

Name	Type	Comment
ParameterNumber	WORD	Number of the parameter
Value	DINT	New value of the specified parameter

10.2.1.3.2.10 MCA_Pos_Ref (STRUCT)

Used for MCA_Indexing

InOut:

Name	Type	Comment
Position	LREAL	[u] Reference position.
Velocity	LREAL	[u/s] Value of the maximum velocity (not necessarily reached)
Acceleration	LREAL	[u/s ²] Value of the acceleration (increasing energy of the motor)
Deceleration	LREAL	[u/s ²] Value of the deceleration (decreasing energy of the motor)
Jerk	LREAL	[u/s ³] Value of the Jerk
Mode	BOOL	TRUE = absolute, FALSE = relative

10.2.1.3.2.11 MC_Cam_Id (STRUCT)

Modification for cam table movement.

Some additional parameter can be provided by accessing the variables in this data structure, after the cam table has been processed by MC_CamTableSelect

InOut:

Name	Type	Comment
MasterPeriodic_Distance	LREAL	Applicable in case the master is not a modulo axis, the CAM movement will be repeated using this distance
SlavePeriodic_Distance	LREAL	Applicable in case the slave is not a modulo axis, the CAM movement will be repeated using this distance
freeze_master_backward	BOOL	If the master moves backward, this movement is not used for the CAM. Just if it passes the frozen position in forward direction, the slave will follow! If the positive movement has a certain velocity when passing the "freeze"-point, a velocity jump will happen
ignoreMasterSyncPosition	BOOL	To be used with MC_CamIn, the CamIn will start directly to synchronize, and will sync within MasterStartDistance, starting with the actual positions it is possible to use this option when: (master = standstill, slave = standstill) (master = moving, slave = moving) (master = moving, slave = standstill) in case:

Name	Type	Comment
		master = standstill, slave = moving, the slave axis will come to an abrupt standstill, as it is not possible to determine a velocity scaling

10.2.1.3.2.12 MC_PProfile (STRUCT)

The data type MC_PProfile is used for CamTable. An array has to be defined and provided at MC_CamTableSelect. Several CamTables could be defined and the axis could change between them on the fly. There is no routine of smooth movement from on table to the next so the user has to take care just to switch on appropriate positions



User can utilize CAM editor in Automation builder to generate Cam table (MC_PProfile) automatically. For more details refer to Automation builder help.

InOut:

Name	Type	Initial	Comment
master_position	LREAL		Masterposition of interpolation point. Be careful, the values have to be in ascending order: MC_PProfile_Array[x].master_position < MC_Profile_Array[x+1].master_position
interpolation_point	LREAL		Slaveposition of interpolation point
velocity_ratio	LREAL	0	Velocity value for interpolation point, is just relevant for MCA_POLY5 and MCA_POLY3. It describes the relation of slave- velocity to master-velocity, e.g. velocity_ratio=1 "Master and Slave have the same velocity.
acceleration_ratio	LREAL	0	Acceleration to be reached at this point. It is just relevant for MCA_POLY5. It gives the relation of slave- acceleration according to master acceleration.

10.2.1.3.2.13 MC_TProfile (STRUCT)

This structure is used for time based profile movement, e.g. MC_PositionProfile

InOut:

Name	Type	Comment
interpolation_point	LREAL	Depending on usage, position, velocity or acceleration.
first_derivative	LREAL	Equivalent to velocity on PositionProfile or acceleration on VelocityProfile. Is just relevant on usage of MCA_POLY5 und MCA_POLY3 relevant.
second_derivative	LREAL	Equivalent to acceleration on PositionProfile. Is just relevant with MCA_POLY5.
delta_time	TIME	Time - Distance in ms to previous interpolation point.

10.2.2 MotionControlLoad (Library)

Motion Control library according to PLCOpen definition for Load control functionality. This Library implementation is based on the "PLCOpen Motion Part 6 – Fluid Power Extensions".

This library contains PLC open standard blocks (MC), and Central Motion control (CMC) function blocks. The library also contains visualization for each function block.

This Library must be used along with ABB_MotionControl_AC500 Library which requires a runtime license.

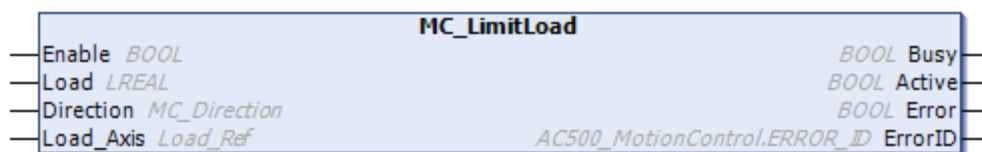
Copyright: We reserve all rights in these programs and the information therein. Reproduction, use or disclosure to third parties without express authority is strictly forbidden. (c) 2006-2021 ABB, all rights reserved

10.2.2.1 01_PLCOpen

PLC open motion control function blocks for load control and torque control.

10.2.2.1.1 MC_LimitLoad (FB)

This function block activates a limitation of the load values provided by an axis.



This may be torque, force, pressure or differential pressure. The measures taken to keep the limits are vendor specific; switching between load and motion control depends on the external load conditions of the axis. The function block sets the 'Busy' output when the limiting measures are stand-by on the axis. The 'Active' output is set, when the limiting measures are active on the axis.





Use Case Rational:

The function block MC_LimitLoad is intended to provide overload protection for a process in terms of driving forces, torque or pressures during motion (e.g. mould protection in injection moulding machines). If load values on the axis exceed the given limit, appro-

appropriate measures are taken to keep this limit, implying that the motion will not be following the programmed path but now depends on the load conditions. However, the 'Active' output of the MC_MoveXXX will stay TRUE in this case, following the modified PLCopen definition "The 'Active' output indicates that the function block has control of the path generation for the axis". This is despite the fact that, physically, only the load-conditions or the movement of an axis can be independently controlled with set values. With actual load below programmed limit, the programmed motion will proceed. The Function block can be applied in different scenarios which could be e.g.

- A more centralized application in terms of a "protection mode", where the complete motion is load limited. In this case the function block would be enabled independently from the motion program itself.

A more decentralized application in terms of additional functionality during the motion program. In this case the function block would be activated by and within the motion program itself. An application example is the mould protection scenario mentioned above, restricting the limiter activity to a certain phase of the programmed motion. Ensuring that limits are only supervised e.g. while one certain MC_MoveXXX has primary control on the axis can be achieved by enabling MC_LimitLoad by the 'Active' output of the MC_MoveXXX. In this way the limitation is only activated when the MC_MoveXXX takes control on the axis for the first time and is deactivated when the MC_MoveXXX loses control on the axis by 'Done', 'CommandAborted' or 'Error'.

	If this block is used during a discrete (positioning) or synchronous movement, the axis will not reach the planned position
	This function block should not be used together with MC_LoadControl function block.
	Issuing MC_LimitLoad does not cause a motion of the axis itself. It is meant to work in parallel to a motion command. It is not guaranteed that activity of the limiting measures will be seen by the function block: a short pulse of the limited quantities could be over before the next Function Block cycle occurs.
	Use just one instance of this block per load axis, because the "Enable" has to define clearly if the functionality is activated. It is not foreseen to interrupt an instance by another of the same type, so the behavior is undefined.

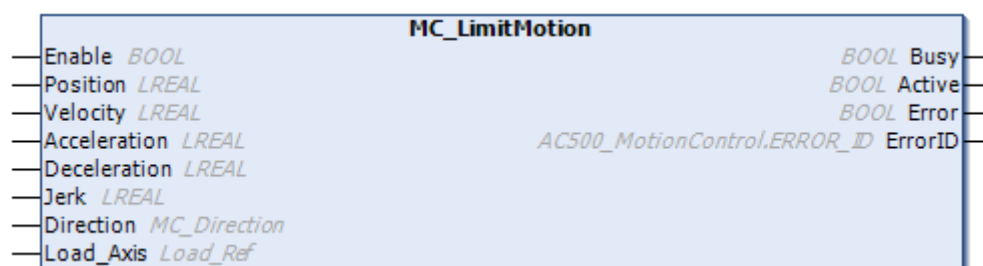
InOut:

Scope	Name	Type	Comment
Input	Enable	BOOL	Allows function block to modify (clamp) a motion command
	Load	LREAL	Value of the maximum applicable load on the axis (Torque, force or pressure in technical unit [u])
	Direction	MC_Direction	Supported direction: Positive, negative, current, default. The direction determines if just positive or negative load values are to be limited, or both (default)

Scope	Name	Type	Comment
Output	Busy	BOOL	The function block is not finished. Output is independent of Axis status
	Active	BOOL	Indicates that the function block has influence on the axis
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	AC500_Motion-Control.ERROR_ID	Error identification. For error details refer to enumeration ERROR_ID from AC500_Motion-Control library
Inout	Load_Axis	Load_Ref	Reference to axis

10.2.2.1.2 MC_LimitMotion (FB)

This function block limits the movement of an axis. Changes in load can result in inexpectant velocity/acceleration deviations, and thus inaccurate limitations






This function block activates a limitation of the motion values of an axis. These are 'Position', 'Velocity', 'Acceleration', 'Deceleration' and 'Jerk'. The measures taken to keep the limits are vendor specific; switching between motion and load control depends on the external load conditions of the axis. The function block sets the 'Busy' output when the limiting measures are standby on the axis. The 'Active' output is set, when the limiting measures are active on the axis.

Use Case Rational

The function block MC_LimitMotion is intended to protect a process from undefined movements during load/torque control. Possible application: e.g. force fitting. The function block is intended to be used in conjunction with a MC_LoadControl or MC_TorqueControl having primary control on the axis. The MC_LimitMotion should be enabled by the 'Active' output of the MC_LoadControl / MC_TorqueControl. If motion values on the axis exceed the given limit, appropriate measures are taken to keep to these limits, implying that the load/torque will not follow the programmed trajectory but depend on the external load conditions. However, the 'Active' output of the MC_LoadControl/MC_TorqueControl will stay TRUE in this case, following the modified PLCOpen definition "The 'Active' output indicates, that the function block has control on the set-value generation of the axis". This is despite the fact, that physically only the load-conditions or the movement of an axis can be controlled. With actual motion states below programmed limits, the programmed load/torque trajectory will proceed. Enabling the limiter block with activation of the MC_LoadControl/MC_TorqueControl ensures that limits are only supervised when the MC_LoadControl/MC_TorqueControl takes control on the

axis for the first time. Disabling the limiter block with de-activation of the MC_LoadControl/MC_TorqueControl ensures that limits are no more supervised when the MC_LoadControl/MC_TorqueControl loses control on the axis by 'CommandAborted' or 'Error'

	It is not guaranteed that activity of the limiting measures will be seen by the function block: a short pulse of the limited quantities could be over before the next Function Block cycle occurs.
	With the given direction CURRENT or DEFAULT, the position will not be limited and the velocity will be limited in both directions. With the given direction POSITIVE, the position will be limited below the given value and the velocity will be limited in both direction. The function block will not start a movement in the opposite direction when the position is already higher when the function block is enabled. It will at maximum reduce the velocity to 0. (same principle behavior in negative direction). The limitations will just be activated when the axis is actually controlled by the LOAD value, e.g. LoadProfile or LoadControl
	Use just one instance of this block per load axis, because the "Enable" has to define clearly if the functionality is activated. It is not foreseen to interrupt an instance by another of the same type, so the behavior is undefined.

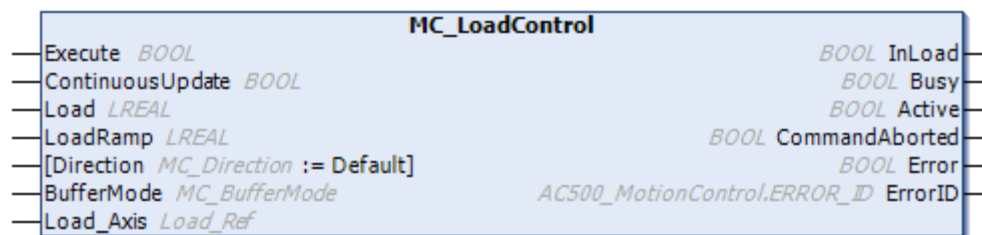
InOut:

Scope	Name	Type	Comment
Input	Enable	BOOL	Allows function block to modify (clamp) a load command
	Position	LREAL	[u] = technical unit. The position limitation is just activated if Direction is Positive or Negative, and will limit the position if reached in the respective direction
	Velocity	LREAL	[u/s] = technical unit. Absolute value of the maximum velocity. Velocity=0 will switch off the velocity limitation
	Acceleration	LREAL	[u/s ²] = technical unit. Value of the maximum acceleration (acceleration is applicable with same sign of torque and velocity) Acceleration = 0 will switch off the acceleration limitation.
	Deceleration	LREAL	[u/s ²] = technical unit. Value of the maximum deceleration (deceleration is applicable with opposite signs of torque and velocity) deceleration = 0 will switch off the deceleration limitation.
	Jerk	LREAL	Value of the maximum jerk (not used)
	Direction	MC_Direction	Supported direction: Positive, negative, current, default. The direction determines if

Scope	Name	Type	Comment
			just positive or negative load values are to be limited, or both (default)
Output	Busy	BOOL	The function block is not finished. Output is independent of Axis status
	Active	BOOL	Indicates that the function block has influence on the axis
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	AC500_MotionControl.ERROR_ID	Error identification. For error details refer to enumeration ERROR_ID from AC500_MotionControl library
Inout	Load_Axis	Load_Ref	Reference to axis

10.2.2.1.3 MC_LoadControl (FB)

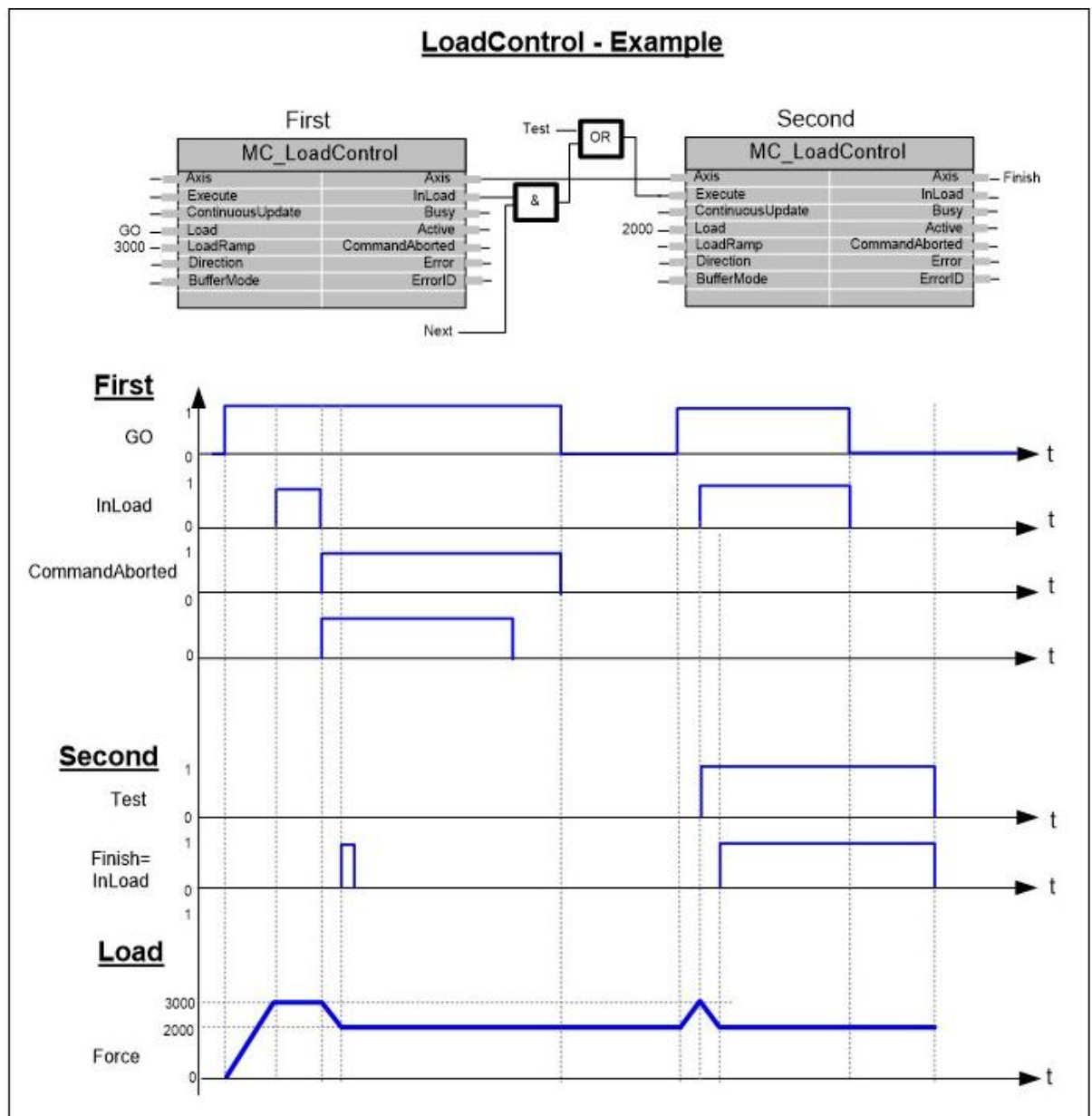
This function block commands a controlled torque/force/pressure movement.



This function block continuously exerts a torque or force or pressure of the specified magnitude. This magnitude is approached using a defined ramp ('LoadRamp'), and the function block sets the 'InLoad' output if the commanded load level is reached. Positive torque, force and differential pressure is in the positive direction of velocity, pressure is physically unsigned.

Example

The function block LoadControl provides base functionality for any application in which the devolution of forces, torque or pressures provided by an axis to a process has to be actively defined and controlled (e.g. in presses).



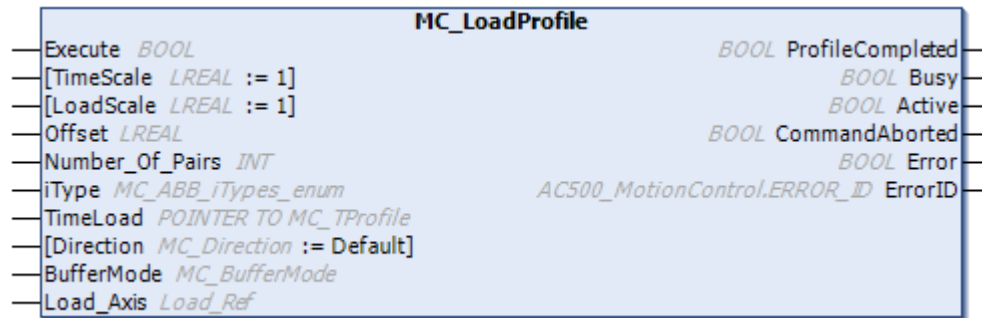
Using this command leads to undefined motion of the axis, unless other (motion controlled) axes or mechanical structures (arrester) are involved.

InOut:

Scope	Name	Type	Initial	Comment
Input	Execute	BOOL		Starts the function block at rising edge
	ContinuousUpdate	BOOL		Decide if new input parameters are processed during the movement. This input is checked only when Execute gets TRUE
	Load	LREAL		[u] = technical unit. Value of the load, Torque/ Force/ Pressure. Positive and negative values are allowed
	LoadRamp	LREAL		[u/s] = technical unit. Absolute value of the input is considered. If LoadRamp = 0, will result in a jump for load- change, as fast as possible
	Direction	MC_Direction	Default	Supported direction: Positive, negative, current, default. The direction determines if just positive or negative load values are to be limited, or both (default)
	BufferMode	MC_BufferMode		Not supported, default mcABORTING used
Output	InLoad	BOOL		Commanded load finally reached, the actual load can be different
	Busy	BOOL		The function block is not finished
	Active	BOOL		Indicates that the function block has control on the axis
	CommandAborted	BOOL		Command is aborted by another command from other PLCopen function block
	Error	BOOL		Signals that error has occurred within function block
	ErrorID	AC500_MotionControl.ERROR_ID		Error identification. For error details refer to enumeration ERROR_ID from AC500_MotionControl library
Inout	Load_Axis	Load_Ref		Reference to axis



10.2.2.1.4 MC_LoadProfile (FB)

This Function Block commands a time-load locked motion profile. The load in the final element in the profile should be maintained. The state remains 'ContinuousMotion'.



Following points gives more details on using the function block

- An array of MC_TProfile has to be created to hold pairs of Load/time. The address of this array has to be used at the TimeLoad input.
- The values “first_derivative” and “second_derivative” are just used when a polynomial interpolation is applied.
- When the first delta_time value is different from 0, this time is used to approach the first load value. Otherwise, the first value is used as a jump.

	This functionality does not mean it runs one profile over and over again: it can switch between different profiles.
	When ProfileCompleted = TRUE (profile is completed), Axis will run with the last load value.

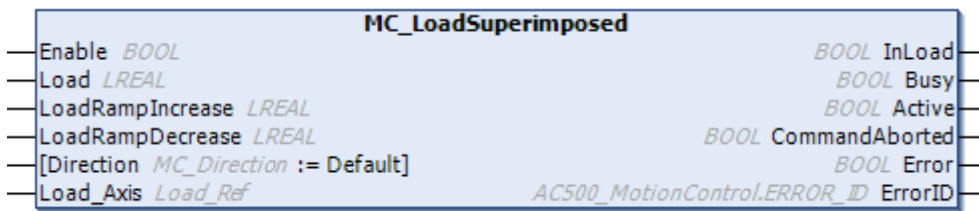
InOut:

Scope	Name	Type	Initial	Comment
Input	Execute	BOOL		Starts the function block at rising edge. The axis will be controlled with the last load value from the profile, and be kept in load control when finished.
	TimeScale	LREAL	1	Scaling for time value in profile. Range: TimeScale > 0
	LoadScale	LREAL	1	Scaling for load value in profile. Range: LoadScale <> 0
	Offset	LREAL		[u] = technical unit. Overall offset for torque, force or pressure profile
	Number_Of_Pairs	INT		Number of sampling points (time/load). Value should be less than or equal to Array size of TimeLoad input. Range: Number_Of_Pairs > 2

Scope	Name	Type	Initial	Comment
	iType	MC_ABB_iTypes_enum		Type of interpolation. Possible values are: MCA_SPLINE_COMPLETE, MCA_SPLINE_NATURAL, MCA_POLY5, MCA_POLY3, MCA_LINEAR
	Time-Load	POINTER TO MC_TProfile		Reference to time/load. MC_TProfile is an ABB specific data type
	Direction	MC_Direction	Default	Supported direction: Positive, negative, current, default. The direction determines if just positive or negative load values are to be limited, or both (default)
	Buffer-Mode	MC_BufferMode		Not supported, default mcABORTING used
Output	ProfileCompleted	BOOL		Profile completed, Commanded position finally reached
	Busy	BOOL		The function block is not finished
	Active	BOOL		Indicates that the function block has control on the axis
	Command-Aborted	BOOL		Command is aborted by another command from other PLCopen function block
	Error	BOOL		Signals that error has occurred within function block
	ErrorID	AC500_MotionControl.ERROR_ID		Error identification. For error details refer to enumeration ERROR_ID from AC500_MotionControl library
Inout	Load_Axis	Load_Ref		Reference to axis

10.2.2.1.5 MC_LoadSuperimposed (FB)

This Function Block commands a controlled torque/force/pressure movement.



This function block commands a controlled load update (increase/decrease) of a specified relative value additional to an existing load. The existing load control operation is not interrupted, but is superimposed by the additional load.

- If MC_LoadSuperImposed is 'Active', then any other command in aborting mode except MC_LoadSuperImposed will abort both load commands: both the MC_LoadSuperImposed and the underlying load command. In any other mode, the underlying load command is not aborted
- If MC_LoadSuperImposed is 'Active' and another MC_LoadSuperImposed is commanded, only the ongoing MC_LoadSuperImposed command is aborted, and replaced by the new MC_LoadSuperImposed command, but not the underlying load command
- The values of 'LoadRampIncrease' and 'LoadRampDecrease' are additional values to the ongoing load control, and not absolute ones. With this, the underlying function block always finishes its job in the same period of time regardless of whether a MC_LoadSuperImposed function block takes place concurrently.
- The output 'Active' has a different behavior as in buffered function blocks.
- The load control will be activated when the axis is on position control at the start of this block. Any ongoing positioning movement will be interrupted.

Use Case Rational:

The function block MC_LoadSuperImposed is intended to allow a superimposed load command to be issued on top of an existing load command without superceding the original load command. If not a load command but a position command is active when MC_LoadSuperimposed is enabled, the ongoing movement will be aborted and the axis be switched to load control. In case that no load control is active when the block is enabled, it will take the actual load value and increase or decrease the load starting from this. When disabled, it will return to this value.

Possible Application:

Actuator: hydraulic cylinder with fluid pressure sensor actuates the press of plastic injection molding machine in a continuous load operation.

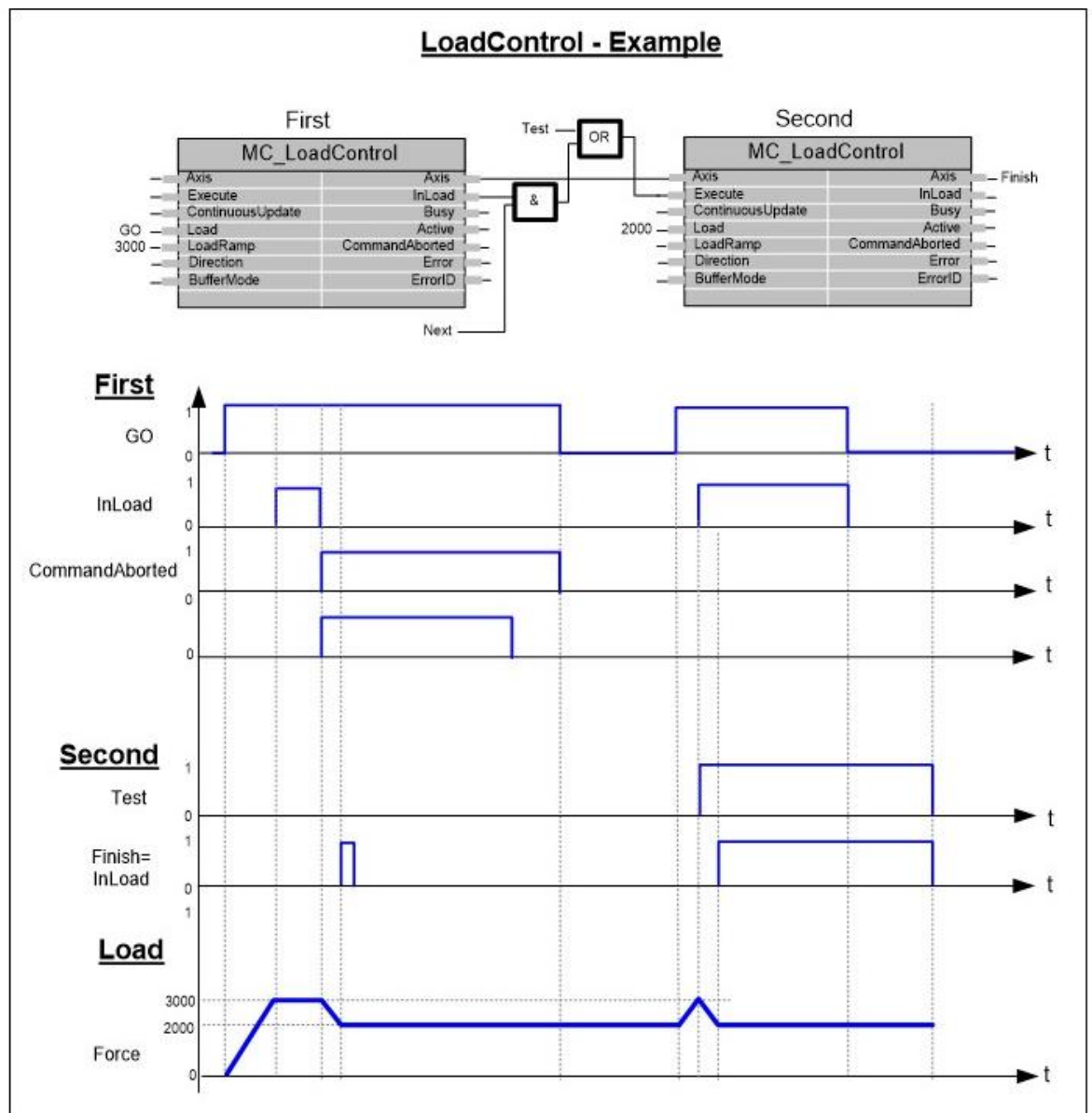
Request: prior to MC_LoadSuperImposed call, a MC_LoadControl block is 'Active' with a command of 7,500 kPa to press melted plastic into the mold. Once the MC_LoadControl

'InLoad' condition is achieved a superimposed pressure of 5,000 kPa is added several times to cause a hammering effect to relieve stresses in the plastic.

Result

The MC_LoadControl pressure command of 7,500 kPa is superimposed with a discrete pressure command of 5,000 kPa. Once the 'LoadSuperImposed' command is active the system pressure rises to 12,500 kPa. When the superimposed pressure command has been achieved the MC_LoadSuperImposed block is done and the original command given by the MC_LoadControl resumes the original pressure command.

The MC_LoadSuperImposed block is executed several times without affecting the original pressure command given by the MC_LoadControl block.



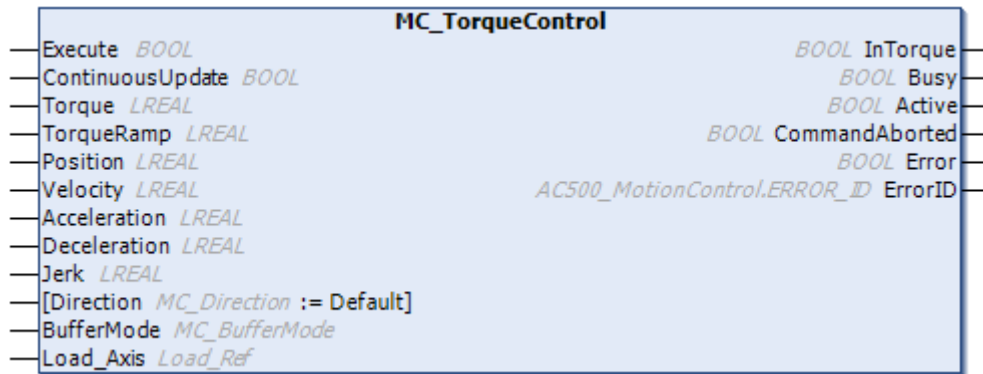
Busy and Active will remain TRUE after function block is disabled until Load is ramped down.

InOut:

Scope	Name	Type	Initial	Comment
Input	Enable	BOOL		Activate the Motion when Enable = TRUE. The axis will be controlled to Load=0 with falling edge, and be kept in load control.
	Load	LREAL		[u] = technical unit. Load that is to be superimposed. Torque/force/pressure
	LoadRampIncrease	LREAL		[u/s] = technical unit. Value of the load ramp increase of the additional load
	LoadRampDecrease	LREAL		[u/s] = technical unit. Value of the load ramp decrease of the additional load
	Direction	MC_Direction	Default	Supported direction: Positive, negative, current, default. The direction determines if just positive or negative load values are to be limited, or both (default)
Output	InLoad	BOOL		Commanded load finally reached
	Busy	BOOL		The function block is not finished
	Active	BOOL		Indicates that the function block has control on the axis
	CommandAborted	BOOL		Command is aborted by another command from other PLCOpen function block
	Error	BOOL		Signals that error has occurred within function block
	ErrorID	AC500_MotionControl.ERROR_ID		Error identification. For error details refer to enumeration ERROR_ID from AC500_MotionControl library
Inout	Load_Axis	Load_Ref		Reference to axis

10.2.2.1.6 MC_TorqueControl (FB)

This Function Block commands a controlled torque/force/pressure movement and also limits the movement of an axis.

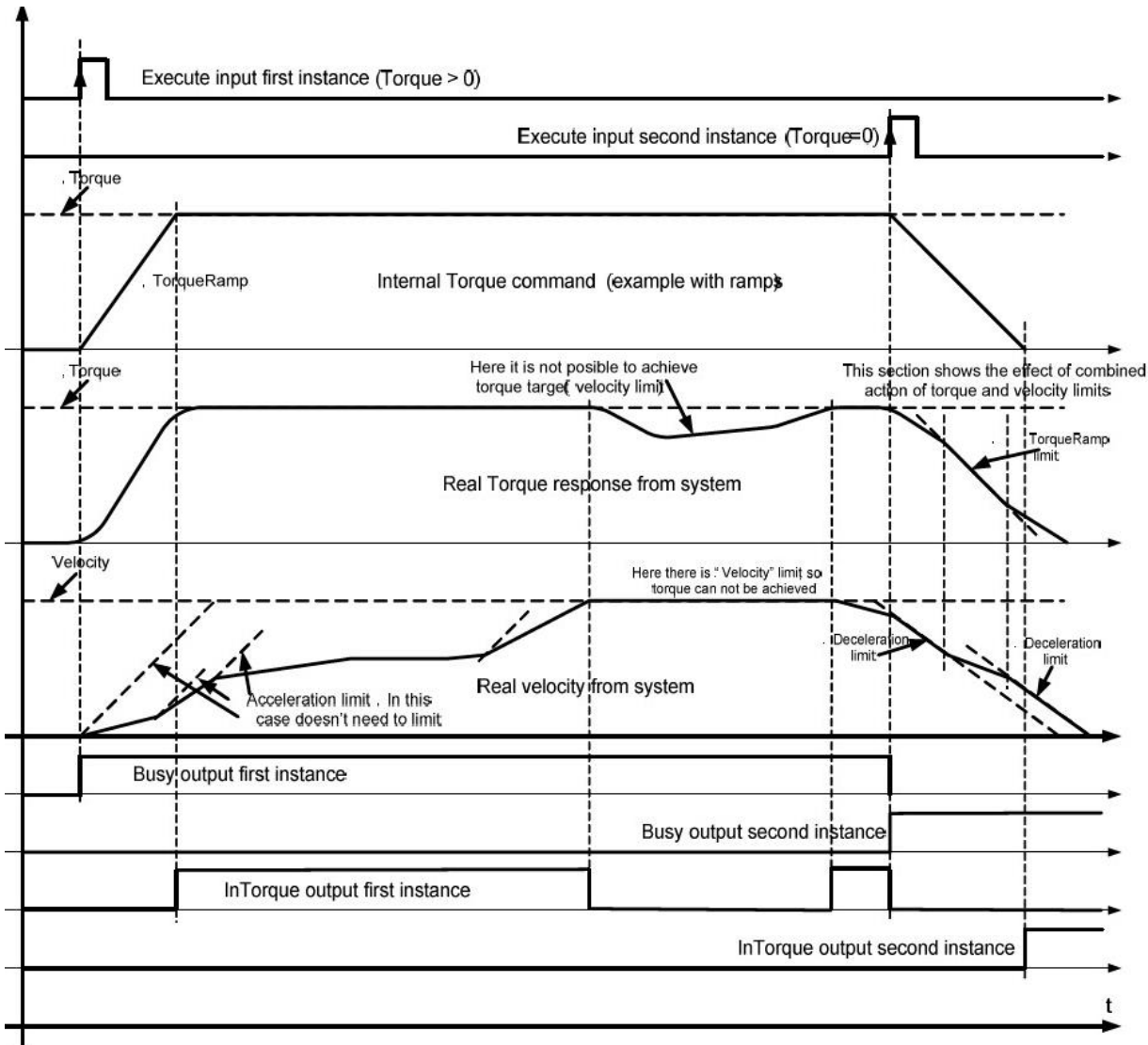


This Function Block continuously exerts a torque or force of the specified magnitude. This magnitude is approached using a defined ramp ('TorqueRamp'), and the Function Block sets the 'InTorque' output if the commanded torque level is reached. This function block is applicable for force and torque. When there is no external load, force is applicable. Positive torque is in the positive direction of velocity.

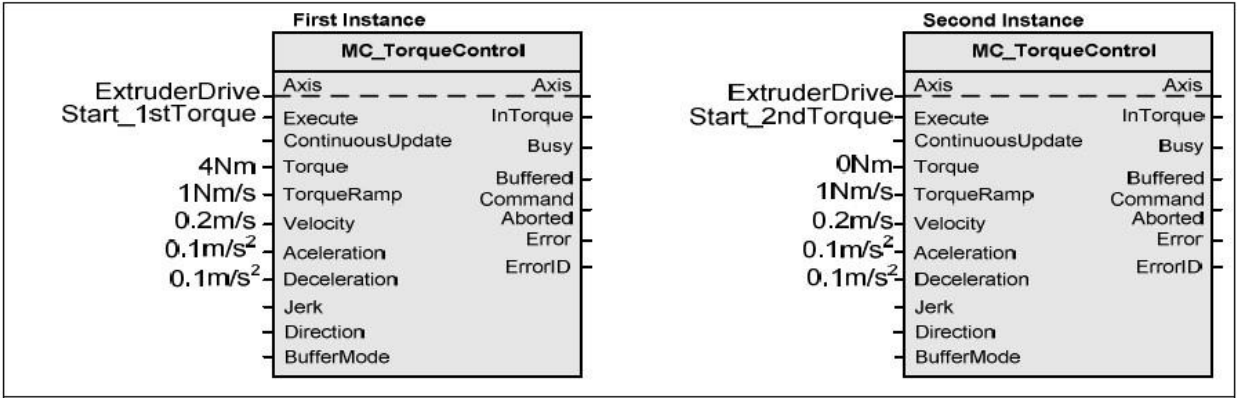
- The movement is limited by velocity, acceleration / deceleration or by the value of the torque.
- Specific additional tests are outside this function block. For instance, checking on the traveled distance could be done via tracing the actual positions during the action.

Example

The example below shows the typical behavior of an intermediate "resistive" load (see 'Deceleration' limit) with some "inertia" (see 'TorqueRamp' limit).



This example could be implemented in a function block diagram like:



This function should not be use together with MC_LimitMotion for the same axis. This function block is implemented as wrapper of two blocks MC_LimitMotion and MC_LoadControl

InOut:

Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
	ContinuousUpdate	BOOL	Decide if new input parameters are processed during the movement. This input is checked only when Execute gets TRUE
	Torque	LREAL	[u] = technical unit. Value of the Torque or Force or Pressure. Positive and negative values are allowed
	TorqueRamp	LREAL	[u/s] = technical unit. Absolute value of the input is considered. If TorqueRamp = 0, will result in a jump for load-change, as fast as possible
	Position	LREAL	[u] = technical unit. The position limitation is just activated if Direction is Positive or Negative, and will limit the position if reached in the respective direction
	Velocity	LREAL	[u/s] = technical unit. Absolute value of the maximum velocity. Velocity=0 will switch off the velocity limitation
	Acceleration	LREAL	[u/s ²] = technical unit. Value of the maximum acceleration (acceleration is applicable with same sign of torque and velocity) Acceleration=0 will switch off the acceleration limitation.
	Deceleration	LREAL	[u/s ²] = technical unit. Value of the maximum deceleration (deceleration is applicable with opposite signs of torque and velocity) deceleration=0 will switch off the deceleration limitation.
	Jerk	LREAL	Value of the maximum jerk (not used)
	Direction	MC_Direction	Supported direction: Positive, negative, current, default. The direction determines if just positive or negative load values are to be limited, or both (default)
	BufferMode	MC_BufferMode	Not supported, default mcABORTING used
Output	InTorque	BOOL	Commanded load finally reached, the actual load can be different

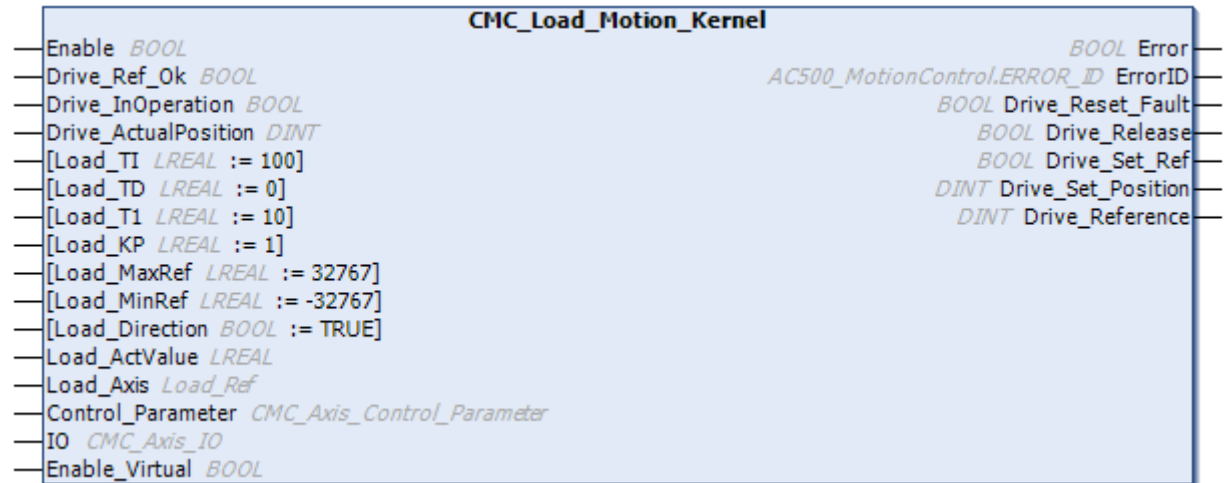
Scope	Name	Type	Comment
	Busy	BOOL	The function block is not finished
	Active	BOOL	Indicates that the function block has control on the axis
	CommandAborted	BOOL	Command is aborted by another command from other PLCopen function block
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	AC500_Motion-Control.ERROR_ID	Error identification. For error details refer to enumeration ERROR_ID from AC500_MotionControl library
Inout	Load_Axis	Load_Ref	Reference to axis

10.2.2.2 CMC_Blocks

Central Motion control (CMC) related function blocks for load control.

10.2.2.2.1 CMC_Load_Motion_Kernel (FB)

This block uses the CMC_Basic_Kernel for motion control regarding position and velocity, and extends the functionality to include loadcontrol



The Load_Ref axis can be used with all PLCopen blocks when usually Axis_Ref is used.

In addition, a number of function blocks are available to control/limit specifically for the load.

These function blocks require an axis of type Load_Ref



Enable_Virtual = TRUE: Load_ActValue is not simulated!

Some explanations for the following parameters:

- An internal PIDT1 control is used for load control
- P - proportional term
- I - integral term
- D - derivative term
- T1 - first order delay for derivative term
- Transfer function $U/E = \text{Load_KP} * (1 + 1/(\text{Load_TI} * s) + \text{Load_TD} * S/((1 + \text{Load_T1} * s)))$

The actuator (drive) has to be accessed outside the CMC_Load_Motion_Kernel block. Actual values and reference values might be transferred by a synchronised bus or by I/Os.

- All inputs and outputs of the function block which are named "Drive_xxxx" should be used to connect to the actuator (drive). It does not matter whether this connection is done by fieldbus or by conventional I/Os.
- The Axis structure is used to connect to the PLCopen blocks
- The Load_Axis structure is used to connect the fluid power PLCopen blocks
- The Control_Parameter structure is used for configuration of control loop.
- The IO structure gives a connection to limit or reference switches.



When using this function block you can use the Visualization “MC_VISU_Load_statemachine” to see the State diagram status. For the function block visualization use the “CMC_VISU_FB_Basic_Kernel” visu from ABB_Motion-Control_AC500 library and map to instance of this function block.

InOut:

Scope	Name	Type	Initial	Comment
Input	Enable	BOOL		Release of function block. Enable has to be set before new control parameters are released by CMC_Axis_Control_Parameter.
	Drive_Ref_Ok	BOOL		Indication for homing
	Drive_InOperation	BOOL		Indication that drive is running. The drive is switched on and is active
	Drive_ActualPosition	DINT		Actual Position in increments
	Load_TI	LREAL	100	[ms] Integration time for load control, 0 will switch of the integral term
	Load_TD	LREAL	0	[ms] Time for derivative control, 0 will switch of the derivative term
	Load_T1	LREAL	10	[ms] Time for a first order delay, applied to the derivative term
	Load_KP	LREAL	1	Proportional gain
	Load_MaxRef	LREAL	32767	Max positive reference, used if load control takes over
	Load_MinRef	LREAL	-32767	Max negative reference, used if load control takes over
	Load_Direction	BOOL	TRUE	TRUE => positive movement => increasing load
	Load_ActValue	LREAL		Measured value for load or torque

Scope	Name	Type	Initial	Comment
Inout	Load_Axis	Load_Ref		Reference to axis
	Control_Parameter	CMC_Axis_Control_Parameter		Parameters for configuration and adjustment of the control loop
	IO	CMC_Axis_IO		By the structure IO (CMC_Axis_IO), some binary inputs are provided. The PLC program has to define a variable of type CMC_Axis_IO and to assign the inputs.
Input	Enable_Virtual	BOOL		Use the axis as virtual axis. Block inputs which are usually received from the real axis are ignored. Required values are generated internally. Enable_Virtual = TRUE: Load_ActValue is not simulated
Output	Error	BOOL		Signals that an error has occurred within the function block.
	ErrorID	AC500_MotionControl.ERROR_ID		Error identification. For error details refer to enumeration ERROR_ID from AC500_MotionControl library
	Drive_Reset_Fault	BOOL		Binary signal to be used for resetting the drive error, if applicable
	Drive_Release	BOOL		Activate the drive
	Drive_Set_Ref	BOOL		Activate homing
	Drive_Set_Position	DINT		Position to be used at homing
	Drive_Reference	DINT		A speed reference

10.2.2.3 Data types

Data type and structures required for the library

10.2.2.3.1 CMC_Load_InOut (STRUCT)

This structure should handle the in-out values to/from the PLCopen-motion function blocks to the internal motion software

InOut:

Name	Type	Comment
Load	LREAL	
LoadRamp	LREAL	[u] torque/force/pressure
SuperImp	CMC_LOAD_SUPER	[u/s] Load Ramp value
LimitLoadValue	LREAL	Superimposed
LimitLoadDirection	MC_DIRECTION	[u] torque/force/pressure
limitLoadActivate	BOOL	Positive,negative, current, default
LimitLoadActivationNumber	DWORD	Limit Load is Activated
LimitMotionPosition	LREAL	Limit Load activation number
LimitMotionVelocity	LREAL	MotionLimitation
LimitMotionAcceleration	LREAL	Limit Load velocity
LimitMotionDeceleration	LREAL	Limit Load acceleration
LimitMotionDirection	MC_DIRECTION	Limit Load deceleration
limitMotionActivate	BOOL	Positive,negative, current, default
LimitMotionActivationNumber	DWORD	Limit Motion activated

Name	Type	Comment
tp_ref	POINTER TO zCMC_TProfile_TABLE	Limit motion activation number
profile_scale	zCMC_AXIS_PROFILE_SCALE	TProfile table reference
start_mode	zCMC_AXIS_START_MODES	Profile scaling
actual_load	LREAL	Start mode
active	BOOL	Actual load
direction	MC_Direction	Activate
isActive	DWORD	Direction
start	BOOL	Is Active
stop	BOOL	Start
in_limit_positive	BOOL	Stop
in_limit_negative	BOOL	Limit is positive
in_motion_limit	BOOL	Limit is negative
isActive_LoadControl	BOOL	Motion Limit
start_confirmation	BOOL	Active Load control
in_load	BOOL	Start is confirmed
load_profile_ready	BOOL	In Load
start_level	BOOL	Load Profile is ready
start_mode_error	BOOL	Start Level

10.2.2.3.2

10.2.2.3.3 CMC_Load_Super (STRUCT)

InOut:

Name	Type	Comment
Load	LREAL	
LoadRampIncrease	LREAL	[u] torque/force/pressure
LoadRampDecrease	LREAL	[u/s] Load ramp increase value
isActive	DWORD	[u/s] Load ramp decrease value
start	BOOL	Is active
stop	BOOL	Start
in_load	BOOL	Stop
active	BOOL	In Load
start_level	BOOL	Active
start_confirmation	BOOL	Start level

10.2.2.3.4 Load_Ref (STRUCT)

Holds the main information regarding the axis. It is used to identify a load axis and connect the various PLCOpen blocks and KERNEL block which belong to the specific load axis.

InOut:

Name	Type	Comment	Inherited from
user	CMC_Axis_User	User Axis details	Axis_Ref
inout	CMC_Axis_InOut	This structure should handle the in-out values to/from the open-motion function blocks to the fieldbus or the internal motion software. It represents a neutral interface	Axis_Ref
actual	CMC_Axis_Actual	Some actual values like positions and velocity per cycle	Axis_Ref
parameter	Axis_Parameter	Parameter which are written/read with MC_WriteParameter/MC_ReadParameter blocks and also parameter used for position control loop	Axis_Ref
load_inout	CMC_Load_InOut		

10.2.2.4 Visualizations

Function block and state machine visualizations

10.2.3 MotionControlEco (Library)

The library provides function blocks to use the PWM and PTO of the eCoV3 PLC to realize the motion control applications.

Along with the kernel function blocks in this library user can use the MC, MCA function blocks from the AC500_MotionControl Library to realize various motion control functionalities like camming, homing and so on. For more details of the available function blocks refer to ABB_MotionControl_AC500 library.

Refer the Automation Builder online help for details on the eCoV3 PLC configuration of PWM, PTO and other limitations.

Copyright: We reserve all rights in these programs and the information therein. Reproduction, use or disclosure to third parties without express authority is strictly forbidden.
(c) 2006-2021 ABB, all rights reserved

10.2.3.1 Data types

Structures related to PTO and PWM configuration.

10.2.3.1.1 OBIO_PTO_Motion_Parameter (STRUCT)

Parameters to be configured for the PTO Motion Control application

InOut:

Name	Type	Initial	Comment
Channel	BYTE	0	0..1, refers to PTO Channels of eCo V3 and requires the outputs to be configured accordingly
Cw_Ccw	BOOL	FALSE	Default: pulse and Direction
Invert_Direction	BOOL	FALSE	With Create_PTO_Position=TRUE: invert Direction for the complete axis. otherwise: invert just Direction for PTO
Precise	BOOL		Add a pulse movement to reach exactly the required position. If min_frequency is too high to control it from PLC program
Create_PTO_Position	BOOL	TRUE	If TRUE, use number of pto pulses as actual position, if FALSE, actual position has to be provided as input DRIVE_ACTUALPOSITION
Min_Frequency	DWORD	400	Define a low frequency limit, to avoid a too slow acceleration start. Deviation with max frequency is 1%, slowest possible frequency is max/625, deviation for slowest frequency is 0.002%. The max frequency is derived from CMC_AXIS_CONTROL_PARAMETER.

10.2.3.1.2 OBIO_PWM_Motion_Parameter (STRUCT)

Parameters to be configured for the PWM Motion Control application

InOut:

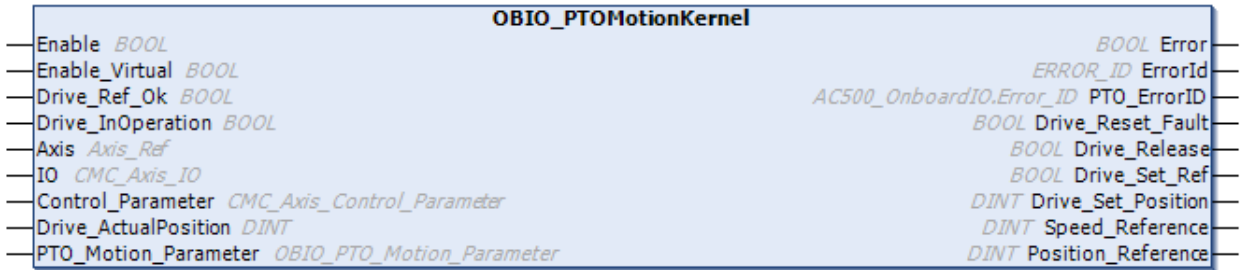
Name	Type	Initial	Comment
Channel	BYTE	0	0..7, refers to PWM Channels of eCo V3 and requires the outputs to be configured accordingly. We recommend to use channels which has faster PWM
PosDirectionState	BOOL		Define if the direction output should have a high (TRUE) or low state to move forward
Create_PWM_Position	BOOL	TRUE	If TRUE, use number of PWM pulses as actual position, if FALSE, actual position has to be provided as input DRIVE_ACTUALPOSITION
DirectionDelay	DINT	10	Define a minimum delay for direction change [ms]

10.2.3.2 eCo Kernel Function blocks

This folder contains the Kernel function blocks required for the PWM and PTO configuration in eCo V3 PLC along with Motion Control Library.

10.2.3.2.1 OBIO_PTOMotionKernel (FB)

This function block extends the basic motion kernel functionality, so that it can be used for the eco PTO outputs and connect a stepper drive.



- 2 axes are possible in total. For each axis one instance of this function block must be called.
- Different parameters related to the PTO can be adjusted in OBIO_PTO_Motion_Parameter
- It is possible to use either the stepper pulses as actual position, or connect a separate value to Drive_ActualPosition

Adjust PTO related parameter in OBIO_PTO_Motion_Parameter

The function block behaves as follows:

- If the input Drive_InOperation is not connected, the required state is created internally.
- If a drive delivers a corresponding signal, the Drive_Release/Drive_InOperation sequence can be used as with CMC_Basic_Kernel.
- If the input Drive_ActualPosition is not connected, the stepper pulses can be used instead.
- The input Drive_ActualPosition can be connected to an external encoder position value. (configure in OBIO_PTO_Motion_Parameter)

The function block connects internally to the PTO outputs. The PTO configuration for the output is required, but no separate function block to use PTO outputs.

PTO parameters and input Enable_Virtual is not recommended to change on the fly or running system. This can lead to unexpected behavior of the system.

This function block supports the visualization “CMC_VISU_FB_Basic_Kernel” from the ABB_MotionControl_AC500 library. User needs to manually map the instance of function block in Visualization reference.

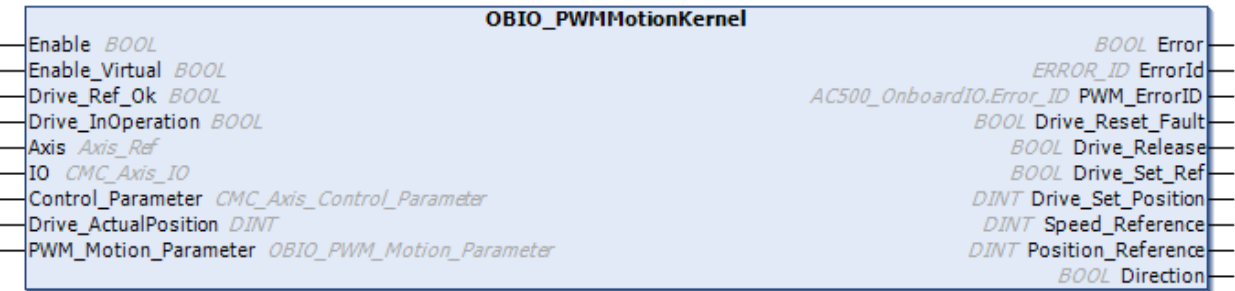
InOut:

Scope	Name	Type	Initial	Comment
Input	Enable	BOOL		Release of function block. Enable has to be set before new control parameters are released by CMC_Axis_Control_Parameter.
	Drive_Ref_Ok	BOOL		Indication for homing
	Drive_InOperation	BOOL		Indication that drive is running. The drive is switched on and is active
	Drive_ActualPosition	DINT		Actual Position in increments
Inout	Axis	Axis_Ref		Reference to the axis to be controlled
	Control_Parameter	CMC_Axis_Control_Parameter		Parameters for configuration and adjustment of the control loop
	IO	CMC_Axis_IO		By the structure IO (CMC_Axis_IO), some binary inputs are provided. The PLC program has to define a variable of type CMC_Axis_IO and to assign the inputs.
Output	Error	BOOL		Signals that an error has occurred within the function block.
	ErrorId	ERROR_ID		<p>The error codes ErrorId also sets the output-bit Error=TRUE and sets the axis in state ERROR_STOP. To allow a new movement the error-codes ErrorId require that either the axis is disabled/enabled by MC_Power or the error reset is disabled/enabled by MC_Reset.</p> <p>The error codes ErrorID_WARNING will not set Error = TRUE, and will not set the axis to ERROR_STOP. The error codes ErrorID_WARNING do not require the MC_Reset or MC_Power. It is possible the axis is stopped and ongoing motion is aborted by a WARNING. The value will be shown until: + An other error or warning occurs + MC_Reset or MC_Power is used</p>
	Drive_Reset_Fault	BOOL		Binary signal to be used for resetting the drive error, if applicable
	Drive_Release	BOOL		Activate the drive
	Drive_Set_Ref	BOOL		Activate homing
	Drive_Set_Position	DINT		Position to be used at homing
	Speed_Reference	DINT		Reference value for the drive
	Position_Reference	DINT		Position reference for the drive in increments

Input	Enable_Virtual	BOOL		Use the axis as virtual axis. Block inputs which are usually received from the real axis are ignored. Required values are generated internally
Output	PTO_ErrorID	AC500_On-boardIO.Error_ID	0	Error code. For error details refer to AC500_OnboardIO library Enum Error_ID
Inout	PTO_Motion_Parameter	OBIO_PTO_Motion_Parameter		Specific parameter set for PTO axis

10.2.3.2.2 OBIO_PWMMotionKernel (FB)

This function block extends the basic motion kernel functionality, so that it can be used for the eco PWM outputs and connect a stepper drive.



- 8 axes are possible in total. We recommend to use the fast PWM axes. Each axis needs one instance of this kernel block called.
- Different parameters related to the PWM can be adjusted in OBIO_PWM_Motion_Parameter
- It is possible to use either the stepper pulses as actual position, or connect a separate value to Drive_ActualPosition

Adjust PWM related parameter in OBIO_PWM_Motion_Parameter

The function block behaves as follows:

- If the input Drive_InOperation is not connected, the required state is created internally.
- If a drive delivers a corresponding signal, the Drive_Release/Drive_InOperation sequence can be used as with CMC_Basic_Kernel.
- If the input Drive_ActualPosition is not connected, the stepper pulses can be used instead.
- The input Drive_ActualPosition can be connected to an external encoder position value. (configure in OBIO_PWM_Motion_Parameter)

	The block connects internally to the PWM outputs. The PWM configuration for the output is required, but no separate FB to use PWM outputs.
	PTO parameters and input Enable_Virtual is not recommended to change on the fly or running system. This can lead to unexpected behavior of the system.
	This function block supports the visualization “CMC_VISU_FB_Basic_Kernel” from the ABB_MotionControl_AC500 library. User needs to manually map the instance of function block in Visualization reference.

InOut:

Scope	Name	Type	Initial	Comment
Input	Enable	BOOL		Release of function block. Enable has to be set before new control parameters are released by CMC_Axis_Control_Parameter.
	Drive_Ref_Ok	BOOL		Indication for homing
	Drive_InOperation	BOOL		Indication that drive is running. The drive is switched on and is active
	Drive_ActualPosition	DINT		Actual Position in increments
Inout	Axis	Axis_Ref		Reference to the axis to be controlled
	Control_Parameter	CMC_Axis_Control_Parameter		Parameters for configuration and adjustment of the control loop
	IO	CMC_Axis_IO		By the structure IO (CMC_Axis_IO), some binary inputs are provided. The PLC program has to define a variable of type CMC_Axis_IO and to assign the inputs.
Output	Error	BOOL		Signals that an error has occurred within the function block.
	ErrorId	ERROR_ID		<p>The error codes ErrorId also sets the output-bit Error=TRUE and sets the axis in state ERROR_STOP. To allow a new movement the error-codes ErrorId require that either the axis is disabled/enabled by MC_Power or the error reset is disabled/enabled by MC_Reset.</p> <p>The error codes ErrorID_WARNING will not set Error = TRUE, and will not set the axis to ERROR_STOP. The error codes ErrorID_WARNING do not require the MC_Reset or MC_Power. It is possible the axis is stopped and ongoing motion is aborted by a WARNING. The value will be shown until: + An other error or warning occurs + MC_Reset or MC_Power is used</p>
	Drive_Reset_Fault	BOOL		Binary signal to be used for resetting the drive error, if applicable
	Drive_Release	BOOL		Activate the drive
	Drive_Set_Ref	BOOL		Activate homing
	Drive_Set_Position	DINT		Position to be used at homing
	Speed_Reference	DINT		Reference value for the drive
	Position_Reference	DINT		Position reference for the drive in increments
Input	Enable_Virtual	BOOL		Use the axis as virtual axis. Block inputs which are usually received from the real axis are ignored. Required values are generated internally

Output	PWM_ErrorID	AC500_OnboardIO.Error_ID	0	Error code. For error details refer to AC500_OnboardIO library Enum Error_ID
	Direction	BOOL		Moving direction, connect this to a binary output
Inout	PWM_Motion_Parameter	OBIO_PWM_Motion_Parameter		Specific parameter set for PWM axis

10.2.4 Ecat_CiA402 (Library)

This library is based on the AC500 system library for EtherCAT. This library will help in Motion control application.

The library has no password protection and is open to be modified to match the required application. For the editable version of library check the example folder. Further functional description of the other parts of the Motion Control Libraries are available in the “AC500_MotionControl” Library.

Copyright: We reserve all rights in these programs and the information therein. Reproduction, use or disclosure to third parties without express authority is strictly forbidden. (c) 2006-2021 ABB, all rights reserved

10.2.4.1 Data Types

10.2.4.1.1 CiA

10.2.4.1.1.1 ECAT_CiA_Object_App (STRUCT)

The values are stored in an array form of type ECAT_CiA_Object_APP.

InOut:

Name	Type	Comment
Index	WORD	Index for SDO Object
Subindex	BYTE	Subindex for SDO Object
DatType	BYTE	Indicates if value is 0 = BYTE, 1 = WORD, 2 = DWORD
Value	DINT	Value to read or write
Read	BOOL	When TRUE, this element will be read when the list is processed by ECAT_Read_Coe_List
Write	BOOL	When TRUE, this element will be written when the list is processed by ECAT_Write_Coe_List
Dummy	WORD	Reserve
Name	STRING	A name for the element, not necessary for the function

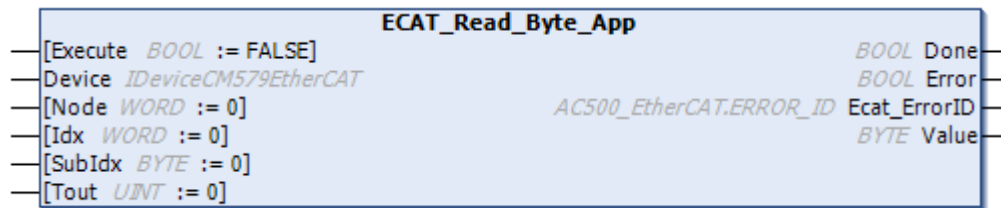
10.2.4.2 POU's

This folder contains the functions and function blocks in the library.

10.2.4.2.1 CoE

10.2.4.2.1.1 ECAT_Read_Byte_App (FB)

The function block reads an 8 bit value from an EtherCAT node. It uses EcatCoeRead to do so.



The differences to “EcatCoeRead” are:

- The block is dedicated to a specific value length
- The value will be received in correct byte order
- The behaviour for Execute, Done, Error corresponds to PLCopen definitions:
 - Execute starts the block with a rising edge
 - The executions ends with output Done or Error
 - Outputs Done and Error are mutually exclusive
 - Outputs Done and Error stay active as long as Execute is TRUE, but for at least 1 cycle when Execute is FALSE



Note: User must make sure correct function block is used to read the value based on the parameter data type. For instance, use ECAT_Read_Byte_App for Parameter with data type BYTE.

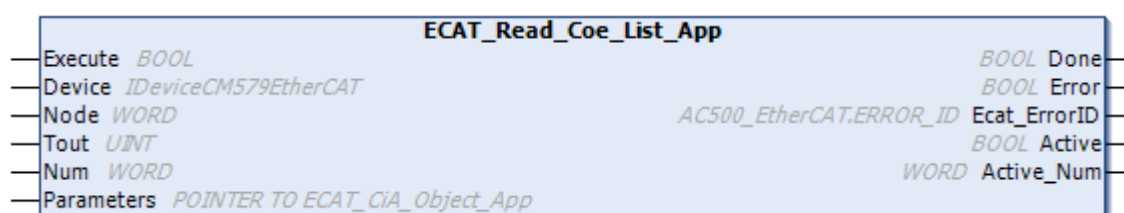
InOut:

Scope	Name	Type	Initial	Comment
Input	Execute	BOOL	FALSE	Starts the function block at rising edge
	Device	IDeviceCM579EtherCAT		Name of the Coupler Device connected, Ex: CM579_ETHCAT
	Node	WORD	0	Node number for EtherCAT device. Ex: 1001, 1002
	Idx	WORD	0	Index of the SDO object
	SubIdx	BYTE	0	Subindex of the SDO object
	Tout	UINT	0	Timeout in milliseconds
Output	Done	BOOL	FALSE	Shows the status of the function block. Done =

				TRUE if the execution is finished
	Error	BOOL	FALSE	Signals that error has occurred within Function block
	Ecat_ErrorID	AC500_EtherCAT.ER-ROR_ID		Error code. Refer ER-ROR_ID from the AC500_ECatBase Library
	Value	BYTE		Value read

10.2.4.2.1.2 ECAT_Read_COE_List_App

The function block reads a list of parameters to the drive by using the EcatCoeRead.



The differences to EcatCoeRead are:

- A list of parameters which are stored in an array could be read automatically and more convenient as it would be by using single blocks.
- The values will be received in correct byte order
- The behaviour for Execute, Done, Error correspond to PLCopen definitions:
 - Execute starts the block with a rising edge
 - The executions ends with output Done or Error
 - Outputs Done and Error are mutually exclusive
 - Outputs Done and Error stay active as long as Execute is TRUE, but for at least 1 cycle when Execute is FALSE

The input Num gives the number of elements in the array to be processed. The output Active_Num shows the actually active element. When an error occurs, the sequence is stopped and the Active_Num indicates which element could not be written.



Note: User must make sure correct function block is used to read the value based on the parameter data type. For instance, use ECAT_Read_Byte_App for Parameter with data type BYTE.

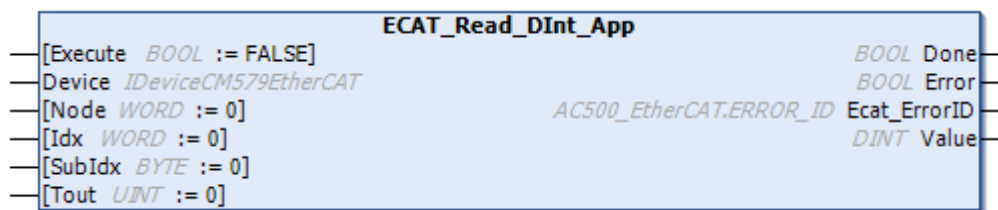
InOut:

Scope	Name	Type	Initial	Comment
Input	Device	IDeviceCM579EtherCAT		Name of the Coupler Device connected, Ex: CM579_ETHCAT
	Node	WORD		Node number for EtherCAT device. Ex: 1001, 1002
	Tout	UNIT		Timeout in milliseconds

	Num	WORD		Number of parameters in the list
	Parameters	POINTER TO ECAT_CiA_Object_App		Address of parameter list
Output	Done	BOOL		Shows the status of the function block. Done = TRUE if the execution is finished
	Error	BOOL	FALSE	Signals that error has occurred within Function block
	Ecat_ErrorID	AC500_EtherCAT.ERROR_ID		Error code. Refer ERROR_ID from the AC500_ECatBase Library
	Active	BOOL		The function block is active
	Active_Num	WORD		Index of parameters which ist actually read

10.2.4.2.1.3 ECAT_Read_DInt_App (FB)

The function block reads a 32 bit value from an EtherCAT node. It uses EcatCoeRead to do so.



The differences to “EcatCoeRead” are:

- The block is dedicated to a specific value length
- The value will be received in correct byte order
- The behaviour for Execute, Done, Error correspond to PLCopen definitions:
 - Execute starts the block with a rising edge
 - The executions ends with output Done or Error
 - Outputs Done and Error are mutually exclusive
 - Outputs Done and Error stay active as long as Execute is TRUE, but for at least 1 cycle when Execute is FALSE



User must make sure correct function block is used to read the value based on the parameter data type. For instance, use ECAT_Read_Byte_App for Parameter with data type BYTE.

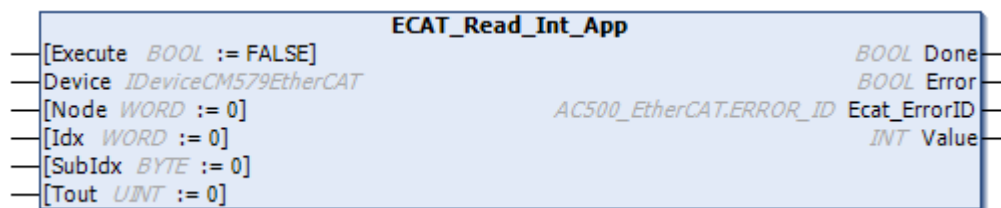
InOut:

Scope	Name	Type	Initial	Comment
Input	Execute	BOOL	FALSE	Starts the function block at rising edge

	Device	IDeviceCM579EtherCAT		Name of the Coupler Device connected, Ex: CM579_ETHCAT
	Node	WORD	0	Node number for EtherCAT device. Ex: 1001, 1002
	Idx	WORD	0	Index of the SDO object
	SubIdx	BYTE	0	Subindex of the SDO object
	Tout	UINT	0	Timeout in milliseconds
Output	Done	BOOL	FALSE	Shows the status of the function block. Done = TRUE if the execution is finished
	Error	BOOL	FALSE	Signals that error has occurred within Function block
	Ecat_ErrorID	AC500_EtherCAT.ERROR_ID		Error code. Refer ERROR_ID from the AC500_ECatBase Library
	Value	DINT		Value read
	Node	WORD	0	Node number for EtherCAT device. Ex: 1001, 1002

10.2.4.2.1.4 ECAT_Read_Int_App (FB)

The function block reads a 16 bit value from an EtherCAT node. It uses EcatCoeRead to do so.



The differences to “EcatCoeRead” are:

- The block is dedicated to a specific value length
- The value will be received in correct byte order
- The behaviour for Execute, Done, Error correspond to PLCopen definitions:
 - Execute starts the block with a rising edge
 - The executions ends with output Done or Error
 - Outputs Done and Error are mutually exclusive
 - Outputs Done and Error stay active as long as Execute is TRUE, but for at least 1 cycle when Execute is FALSE



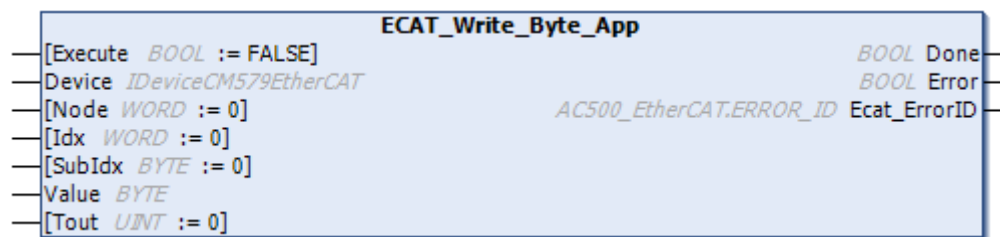
User must make sure correct function block is used to read the value based on the parameter data type. For instance, use ECAT_Read_Byte_App for Parameter with data type BYTE.

InOut:

Scope	Name	Type	Initial	Comment
Input	Execute	BOOL	FALSE	Starts the function block at rising edge
	Device	IDeviceCM579EtherCAT		Name of the Coupler Device connected, Ex: CM579_ETHCAT
	Node	WORD	0	Node number for EtherCAT device. Ex: 1001, 1002
	Idx	WORD	0	Index of the SDO object
	SubIdx	BYTE	0	Subindex of the SDO object
	Tout	UINT	0	Timeout in milliseconds
Out-put	Done	BOOL	FALSE	Shows the status of the function block. Done = TRUE if the execution is finished
	Error	BOOL	FALSE	Signals that error has occurred within Function block
	Ecat_ErrorID	AC500_EtherCAT.ERROR_ID		Error code. Refer ERROR_ID from the AC500_ECat-Base Library
	Value	INT		Value read

10.2.4.2.1.5 ECAT_Write_Byte_App (FB)

The function block writes an 8 bit value to an EtherCAT node. It uses EcatCoeWrite to do so.



The differences to “EcatCoeWrite” are:

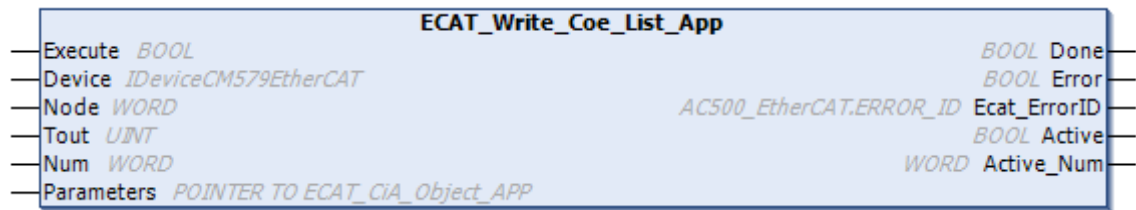
- The block is dedicated to a specific value length
- The value will be written in correct byte order
- The behaviour for Execute, Done, Error correspond to PLCopen definitions:
 - Execute starts the block with a rising edge
 - The executions ends with output Done or Error
 - Outputs Done and Error are mutually exclusive
 - Outputs Done and Error stay active as long as Execute is TRUE, but for at least 1 cycle when Execute is FALSE

InOut:

Scope	Name	Type	Initial	Comment
Input	Execute	BOOL	FALSE	Starts the function block at rising edge
	Device	IDeviceCM579EtherCAT		Name of the Coupler Device connected, Ex: CM579_ETHCAT
	Node	WORD	0	Node number for EtherCAT device. Ex: 1001, 1002
	Idx	WORD	0	Index of the SDO object
	SubIdx	BYTE	0	Subindex of the SDO object
	Value	BYTE		Value to write
Output	Tout	UINT	0	Timeout in milliseconds
	Done	BOOL	FALSE	Shows the status of the function block. Done = TRUE if the execution is finished
	Error	BOOL	FALSE	Signals that error has occurred within Function block
	Ecat_ErrorID	AC500_EtherCAT.ERROR_ID		Error code. Refer ERROR_ID from the AC500_ECatBase Library

10.2.4.2.1.6 ECAT_Write_Coe_List_App (FB)

The function block writes a list of parameters to the drive by using the EcatCoeWrite.



The differences to EcatCoeWrite are:

- A list of parameters which are stored in an array could be written automatically and more convenient as it would be by using single blocks.
- The values will be written in correct byte order
- The behaviour for Execute, Done, Error correspond to PLCopen definitions:
 - Execute starts the block with a rising edge
 - The executions ends with output Done or Error
 - Outputs Done and Error are mutually exclusive
 - Outputs Done and Error stay active as long as Execute is TRUE, but for at least 1 cycle when Execute is FALSE

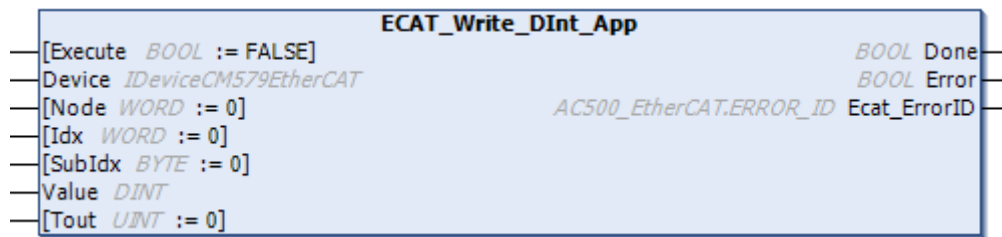
The input Num gives the number of elements in the array to be processed. The output Active_Num shows the actually active element. When an error occurs, the sequence is stopped and the Active_Num indicates which element could not be written.

InOut:

Scope	Name	Type	Initial	Comment
Input	Execute	BOOL		Starts the function block at rising edge
	Device	IDeviceCM579Ether-CAT		Name of the Coupler Device connected, Ex: CM579_ETHCAT
	Node	WORD		Node number for EtherCAT device. Ex: 1001, 1002
	Tout	UINT		Timeout in milliseconds
	Num	WORD		Number of parameters in the list
	Parameters	POINTER TO ECAT_CiA_Object_APP		Address of parameter list
Output	Done	BOOL		Shows the status of the function block. Done = TRUE if the execution is finished
	Error	BOOL	FALSE	Signals that error has occurred within Function block
	Ecat_ErrorID	AC500_EtherCAT.ERROR_ID		Error code. Refer ERROR_ID from AC500_ECatBase Library
	Active	BOOL		The function block is active
	Active_Num	WORD		Index of parameter which is actually written

10.2.4.2.1.7 ECAT_Write_DInt_App (FB)

The function block writes a 32 bit value to an EtherCAT node. It uses EcatCoeWrite to do so



The differences to “EcatCoeWrite” are:

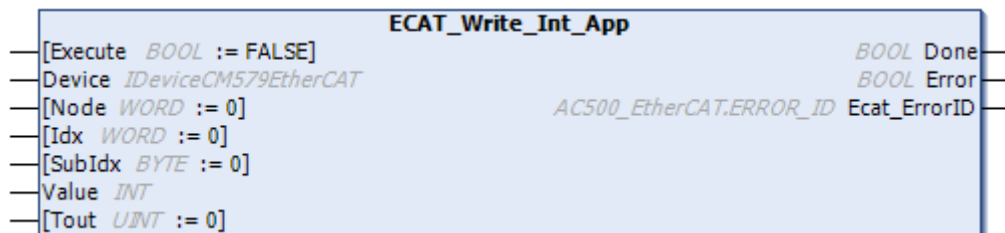
- The block is dedicated to a specific value length
- The value will be written in correct byte order
- The behaviour for Execute, Done, Error correspond to PLCopen definitions:
 - Execute starts the block with a rising edge
 - The executions ends with output Done or Error
 - Outputs Done and Error are mutually exclusive
 - Outputs Done and Error stay active as long as Execute is TRUE, but for at least 1 cycle when Execute is FALSE

InOut:

Scope	Name	Type	Initial	Comment
Input	Execute	BOOL	FALSE	Starts the function block at rising edge
	Device	IDeviceCM579EtherCAT		Name of the Coupler Device connected, Ex: CM579_ETHCAT
	Node	WORD	0	Node number for EtherCAT device. Ex: 1001, 1002
	Idx	WORD	0	Index of the SDO object
	SubIdx	BYTE	0	Subindex of the SDO object
	Value	DINT		Value to write
	Tout	UINT	0	Timeout in milliseconds
Out-put	Done	BOOL	FALSE	Shows the status of the function block. Done = TRUE if the execution is finished
	Error	BOOL	FALSE	Signals that error has occurred within Function block
	Ecat_ErrorID	AC500_EtherCAT.ERROR_ID		Error code. Refer ERROR_ID from the AC500_ECat-Base Library

10.2.4.2.1.8 ECAT_Write_Int_App (FB)

The function block writes a 16 bit value to an EtherCAT node. It uses EcatCoeWrite to do so



The differences to “EcatCoeWrite” are:

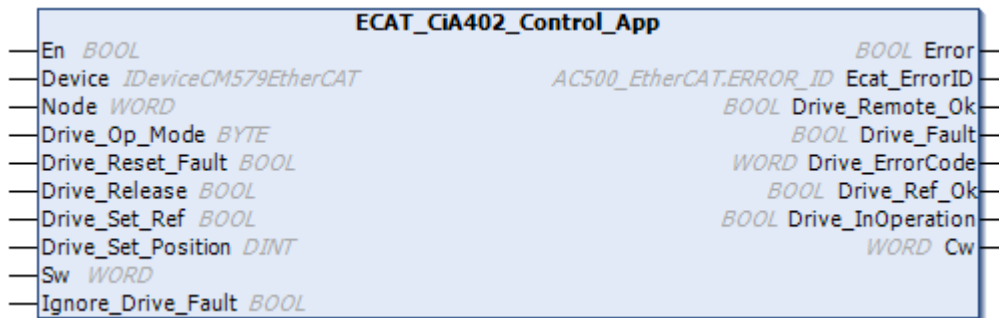
- The block is dedicated to a specific value length
- The value will be written in correct byte order
- The behaviour for Execute, Done, Error correspond to PLCopen definitions:
 - Execute starts the block with a rising edge
 - The executions ends with output Done or Error
 - Outputs Done and Error are mutually exclusive
 - Outputs Done and Error stay active as long as Execute is TRUE, but for at least 1 cycle when Execute is FALSE

InOut:

Scope	Name	Type	Initial	Comment
Input	Execute	BOOL	FALSE	Starts the function block at rising edge
	Device	IDeviceCM579EtherCAT		Name of the Coupler Device connected, Ex: CM579_ETHCAT
	Node	WORD	0	Node number for EtherCAT device. Ex: 1001, 1002
	Idx	WORD	0	Index of the SDO object
	SubIdx	BYTE	0	Subindex of the SDO object
	Value	INT		Value to write
Output	Tout	UINT	0	Timeout in milliseconds
	Done	BOOL	FALSE	Shows the status of the function block. Done = TRUE if the execution is finished
	Error	BOOL	FALSE	Signals that error has occurred within Function block
	Ecat_ErrorID	AC500_EtherCAT.ERROR_ID		Error code. Refer ERROR_ID from the AC500_ECatBase Library

10.2.4.2.2 Drive

10.2.4.2.2.1 ECAT_CiA402_Control_App (FB)



The function block controls the statemachine for a drive with 402 profile and connected to EtherCAT.

- It has to be connected to the Kernel block which controls the statemachine
- It can be connected to Axis_Ref, this gives information to PLCopen blocks about drive problems.

The function block evaluates the status word of a drive and to write the control word.

- En = FALSE: the control word is set to “0”, no command will be accepted
- Drive_Fault = FALSE and Drive_Remote_Ok = TRUE: the drive is ready to be switched on
- Drive_Release: a positive edge will switch on the drive. In case of a fault, the control word will not be modified. A positive edge on drive_release or drive_reset_fault will be necessary to switch on again
- Drive_InOperation = TRUE: the drive is in state “Operation enabled”

The block also has a visualization element which can show the actual state “Visu_ECACiA402_STATE”.

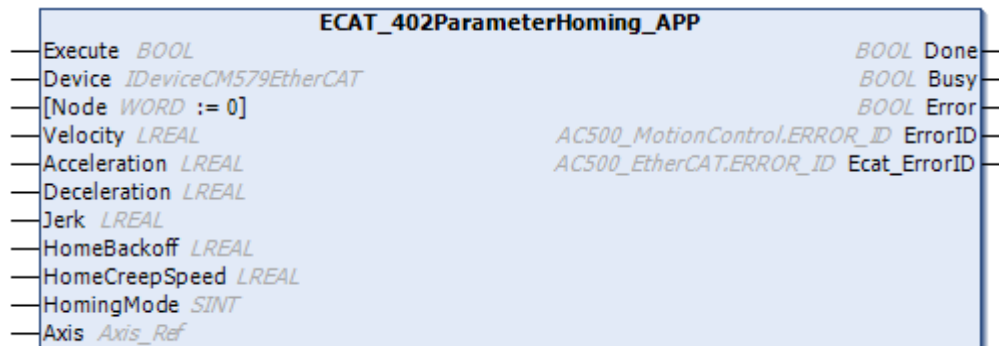
InOut:

Scope	Name	Type	Comment
Input	En	BOOL	Enable the function block
	Device	IDeviceCM579EtherCAT	Name of the Coupler Device connected, Ex: CM579_ETHCAT
	Node	WORD	Node number for EtherCAT device, (1001, 1002...)
	Drive_Op_Mode	BYTE	Use 8 = CSP (position), 9 = CSV (velocity), 0: will not be modified (use a value if drive based homing is required)
	Drive_Reset_Fault	BOOL	Directly mapped to RESET in drive control word
	Drive_Release	BOOL	Switch on the drive by a positive edge
Output	Drive_Set_Ref	BOOL	Activate homing
	Drive_Set_Position	DINT	Position to be used at homing

	Sw	WORD	Statusword from drive
	Ignore_Drive_Fault	BOOL	A drive fault in SW will not change the state machine
	Error	BOOL	Signals that error has occurred within Function block
	Ecat_ErrorID	AC500_EtherCAT.ERROR_ID	Error code. Refer ERROR_ID from the AC500_ECatBase Library
	Drive_Remote_Ok	BOOL	Directly mapped from remote bit in drive status word
	Drive_Fault	BOOL	Directly mapped from fault bit in drive status word, connect with Axis_Ref
	Drive_ErrorCode	WORD	Read from object 16#603F, Connect with Axis_Ref
Output	Drive_Ref_Ok	BOOL	Indication for homing
	Drive_InOperation	BOOL	Drive successfully switched on, state OPERATION_ENABLED
	Cw	WORD	Controlword for the drive

10.2.4.2.3 Homing

10.2.4.2.3.1 ECAT_402ParameterHoming_APP (FB)

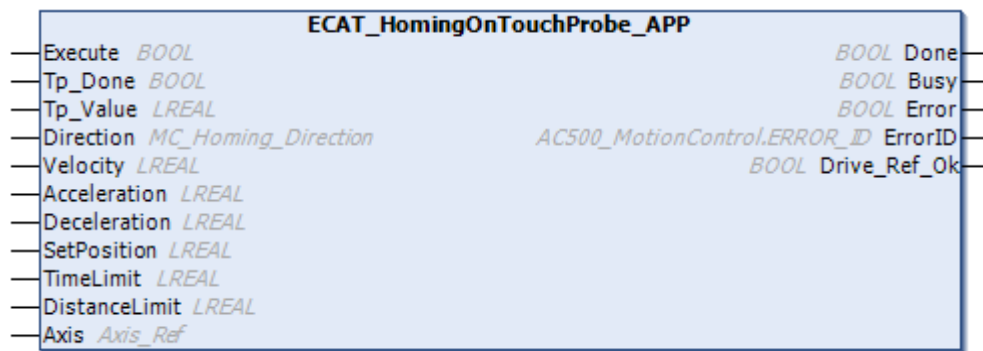


This function block sends the parameter needed for homing to the drive it will send every parameter with a value $\neq 0$

InOut:

Scope	Name	Type	Initial	Comment
Input	Execute	BOOL		Starts the function block at rising edge
	Device	IDeviceCM579EtherCAT		Name of the Coupler Device connected, Ex: CM579_ETHCAT
	Node	WORD	0	Node number for EtherCAT device, (1001, 1002...)
	Velocity	LREAL		Velocity during homing [units/s]
	Acceleration	LREAL		Acceleration during homing [units/s^2]
	Deceleration	LREAL		Deceleration during homing [units/s^2]
	Jerk	LREAL		Jerk during homing [units/s^3]
	HomeBackoff	LREAL		[u] HomeBackoff determines the back-off speed from a home switch
	HomeCreep-Speed	LREAL		[u/s] During the final stage of homing moves, the axis operates at creep speed when locating an index pulse. The creep speed is expressed in user velocity units
	HomingMode	SINT		HomingMode defines the type of homing sequence to be performed when the drive is powered up and then enabled
Output	Done	BOOL		Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BOOL		The function block is active
	Error	BOOL		Signals that error has occurred within Function block
	ErrorID	AC500_MotionControl.ERROR_ID		Shows error number, Refer ERROR_ID enum in AC500_MotionControl library
	Ecat_ErrorID	AC500_EtherCAT.ERROR_ID		Error code. Refer ERROR_ID from the AC500_ECatBase Library
Inout	Axis	Axis_Ref		Reference to axis

10.2.4.2.3.2 ECAT_HomingOnTouchProbe_APP (FB)



This function block performs a homing on a latched position value, e.g. Touch Probe on Z-pulse of an encoder. The execution of the function block will do the following:

- Axis state will change to HOMING
- Axis will move with configured parameters until rising edge of Tp_Done
- Axis will stop, axis state will change to STANDSTILL
- Position value from Tp_Value will be used to set a new position to the axis
- Function block output signal Done will be TRUE as long Execute input signal is TRUE, but at least for one cycle

This function block uses instances of MC_StepRefPulse and MC_SetPosition

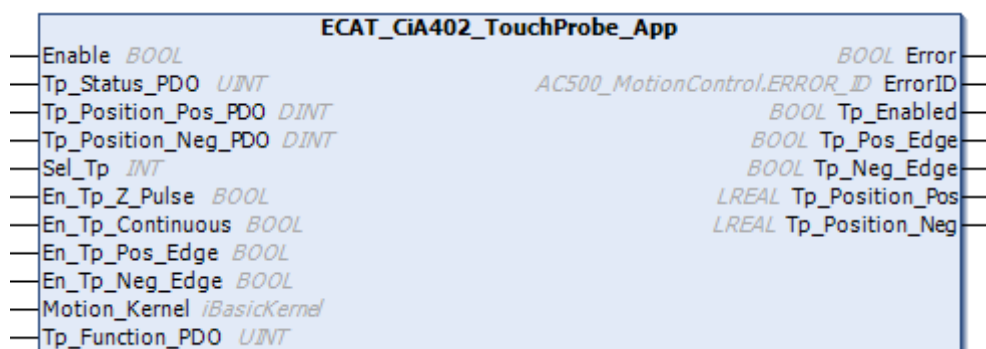
InOut:

Scope	Name	Type	Comment
Input	Execute	BOOL	Starts the function block at rising edge
	Tp_Done	BOOL	Uses rising edge that signals a new latch value is available
	Tp_Value	LREAL	Latch Value, position in [units]
	Direction	MC_Homing_Direction	Specifies the direction to start the homing, just MC_Positive and MC_Negative are possible to use: MC_Positive = Starts in positive direction always MC_Negative = Starts in negative direction always
	Velocity	LREAL	Velocity during homing [units/s]
	Acceleration	LREAL	Acceleration during homing [units/s^2]

Scope	Name	Type	Comment
	Deceleration	LREAL	Deceleration during homing [units/s^2]
	SetPosition	LREAL	New position to be set where latch value occurs [units]
	TimeLimit	LREAL	Time limit to finish homing [s]
	DistanceLimit	LREAL	Distance limit to finish homing [units]
Output	Done	BOOL	Shows the status of the function block. Done = TRUE if the execution is finished
	Busy	BOOL	The function block is active
	Error	BOOL	Signals that error has occurred within function block
	ErrorID	AC500_MotionControl.ERROR_ID	Shows error number, Refer ERROR_ID enum in AC500_MotionControl library
	Drive_Ref_Ok	BOOL	To be connected with input "Drive_Ref_Ok" of the kernel function block
Inout	Axis	Axis_Ref	Connect axis variable here

10.2.4.2.4 TouchProbe

10.2.4.2.4.1 ECAT_CiA402_TouchProbe_App (FB)



This function block manages the Touch Probe objects according to: "EtherCAT Implementation Directive for CiA402 Drive Profile" Document: ETG.6010 D (R) V1.0.0

InOut:

Scope	Name	Type	Comment
Input	Enable	BOOL	Enables processing of function block
	Tp_Status_PDO	UINT	Touch probe status word, corresponds to object "Touch probe status" Index: 0x60B9"
	Tp_Position_Pos_PDO	DINT	Latch value from positive edge from Touch Probe, corresponds to object "Touch probe position positive value" Index: 0x60BA, [increments]
	Tp_Position_Neg_PDO	DINT	Latch value from negative edge from Touch Probe, corresponds to object "Touch probe position negative value" Index: 0x60BB, [increments]
	Sel_Tp	INT	1 = Touch Trobe 1 will be used, 2 = Touch Trobe 2 will be used
	En_Tp_Z_Pulse	BOOL	TRUE = Touch Probe on Z-pulse, FALSE = Touch Probe on "Input"
	En_Tp_Continuous	BOOL	TRUE = Enables continuous mode for Touch Probe, FALSE = enables single mode for Touch Probe
	En_Tp_Pos_Edge	BOOL	TRUE = Enables Touch Probe on positive edge from signal, FALSE = disables Touch Probe on positive edge from signal
	En_Tp_Neg_Edge	BOOL	TRUE = Enables Touch Probe on negative edge from signal, FALSE = disables Touch Probe on negative edge from signal
Output	Error	BOOL	Signals that an error has occurred within function block
	ErrorID	AC500_Motion-Control.ERROR_ID	Shows error number, Refer ERROR_ID enum in AC500_MotionControl library
	Tp_Enabled	BOOL	Touch Probe 1 object is activated
	Tp_Pos_Edge	BOOL	True when new latch value available for Touch Probe on positive edge, stays TRUE in single mode, TRUE for one cycle in continuous mode
	Tp_Neg_Edge	BOOL	True when new latch value available for Touch Probe on negative edge, stays TRUE in single mode, TRUE for one cycle in continuous mode
	Tp_Position_Pos	LREAL	Latch value from positive edge from Touch Probe, [Position output converted to motion control library scaled units]
	Tp_Position_Neg	LREAL	Latch value from negative edge from Touch Probe, [Position output converted to motion control library scaled units]
Inout	Motion_Kernel	CMC_Basic_Kernel	Connect instance name of motion kernel here

Scope	Name	Type	Comment
	Tp_Function_PDO	UINT	Control word for Touch Probe configuration, corresponds to object "Touch probe function" Index 0x60B8

10.2.5 MathFunctions (Library)

This library has mathematical functions like some linear equations, matrix and vector calculations. Current version has only function block for Math linear regression.

Copyright: We reserve all rights in these programs and the information therein. Reproduction, use or disclosure to third parties without express authority is strictly forbidden. (c) 2006-2020 ABB, all rights reserved

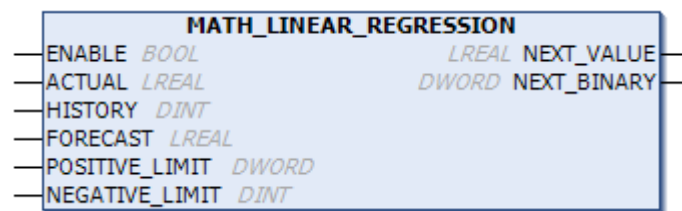
10.2.5.1 POU's

10.2.5.1.1 Math Library

10.2.5.1.1.1 LinearEquation

10.2.5.1.1.2 MATH_LINEAR_REGRESSION (FB)

This function block calculates the estimated next value based on a linear regression with a 8 values history. It uses "least square" GAUSS algorithm



When this function block is used to process an actual position, 2 different purposes are fulfilled:

- A jitter or noise can be compensated
- It is possible to calculate a forecast-position to compensate for a delay in position measurement



Process the actual position or any other master axis always before the slave axis. Otherwise, an additional 1 cycle-delay is introduced

The block calculates the progress for a variable which is captured in equidistant periods of time and is assumed to follow a linear curve. It uses the Gauss "least squares" algorithm to do so. The line is calculated in a way that the sum of squares for the distances from the measured points to the assumed straight line is minimized. A noise or jitter influence of the value is compensated and a predictive value for the variable with an adjustable forecast horizon can be calculated

The gradient and offset for the line are calculated in a way that "sum" is minimized. Then these 2 values are used to calculate the forecast value:

Linear equation:

$$\text{Line}[i] = \text{gradient} * i + \text{offset}$$

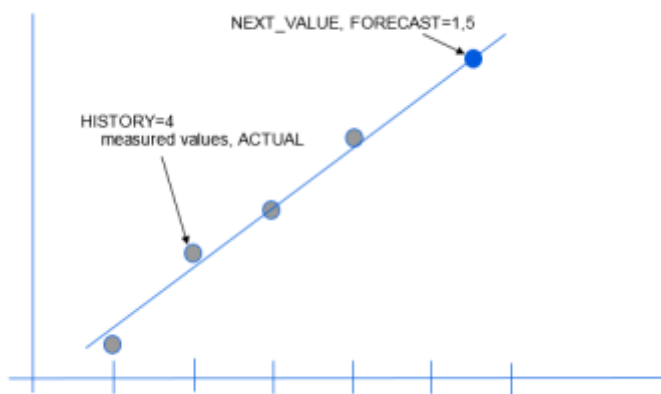
Sum of squares:

$$\text{sum} = \sum_{i=1}^{\text{history}} (x[i] - \text{line}[i])^2$$

FORECAST=0 would mean: value right now, no future or past considered.

$$\text{NEXT_VALUE} := \text{gradient} * \text{FORECAST} + \text{offset}$$

When the ACTUAL value is a modulo value, for example a single turn encoder or a rollover axis, this has to be considered in the calculation. The 2 input values POSITIVE_LIMIT and NEGATIVE_LIMIT can be used to configure this. They define the upper and lower limit for ACTUAL. Also, the NEXT_BINARY will as a result be limited to these borders.



InOut:

Scope	Name	Type	Comment
Input	ENABLE	BOOL	Enable the function block
	ACTUAL	LREAL	Actual value
	HISTORY	DINT	Number of values to be stored to calculate an average
	FORECAST	LREAL	Number of cycles to calculate the future, can be a fractional number
	POSITIVE_LIMIT	DWORD	Use a value >0 in case the input is based on a modulo (rollover) value

Scope	Name	Type	Comment
	NEGATIVE_LIMIT	DINT	Minimum value to be used for a modulo calculation
Output	NEXT_VALUE	LREAL	Result from least square algorithm as LREAL
	NEXT_BINARY	DWORD	Result from least square algorithm as binary value

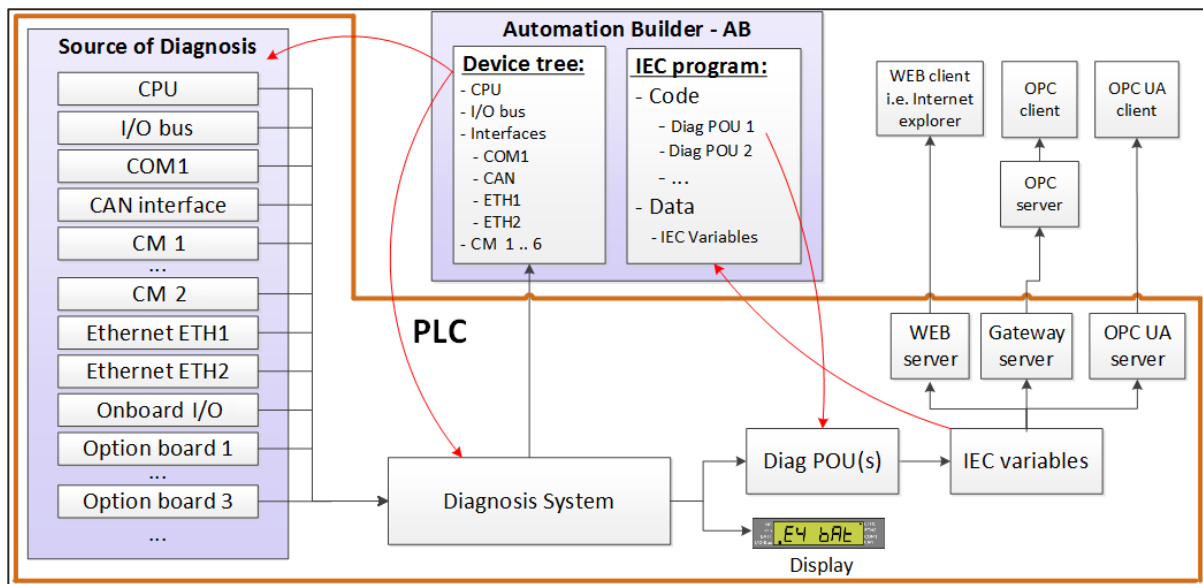
11 DIAGNOSIS

The diagnosis system enables uniform diagnosis of the CPU and its local interfaces, of the local I/O bus with the connected S500 I/O devices and of the fieldbuses connected via communication modules, considering the special features of the various fieldbuses. The safety CPU is also integrated into the diagnosis system.

Diagnosis data of the devices can be accessed by

- CPU display
- Automation Builder
- IEC application

To forward the information to notify them by, e.g., webserver or OPC UA server, the data retrieved in IEC application can be stored in variables



Refer Automation Builder online help chapter PLC Automation with V3 CPUs > Diagnosis and debugging for AC500 V3 products > The diagnosis system for up-to-date information.

11.1 Online diagnosis using Automation Builder

To use the diagnosis system in Automation Builder, login to the online mode is required ↘ “Entering/leaving the online mode”. The online diagnosis in Automation Builder consists of a set of partly animated, mostly read only views. They can be invoked by a double-click on a project tree element which shows a circle indicating that this element is able to show diagnosis messages ↘ “Project tree in online mode”.

Available online diagnosis and statistics:

- Diagnosis messages

When the Automation Builder is switched to online mode, incoming diagnosis messages are displayed as plain-text ↘ “Diagnosis in Automation Builder”.

- CPU/PLC diagnosis
 - “CPU diagnosis views”.
- I/O module diagnosis
 - “Live values in views with I/O components”.
- Communication module and fieldbus diagnosis

- “Communication module and fieldbus diagnosis”
- Diagnosis in IEC application
 - “Diagnosis in IEC application”

For information on the disk status, diagnosis information can be read out with the function blocks PmDiskStatus and PmDiskLifetimeUsed. ↗ “Health monitoring”

Refer Automation Builder online help chapter PLC Automation with V3 CPUs > Diagnosis and debugging for AC500 V3 products > The diagnosis system > Diagnosis in Automation Builder for up-to-date information.

11.2 Diagnosis in PLC program

There are two possibilities for accessing the diagnosis messages in the IEC application:

System diagnosis: Access to diagnosis messages of the whole PLC

Device diagnosis: Access to the diagnosis messages of a device

For both possibilities common data types (structures and enumerations) are defined in the library AC500_DiagTypes ↗ “Data types in library AC500_DiagTypes”. The library is automatically included in PLC project.

Details on how to integrate and use function blocks to receive diagnosis messages from the CPU and fieldbus devices are given in the application examples:

- [AC500 V3 diagnosis in IEC application](#)
- [AC500 V3 diagnosis](#)

Refer Automation Builder online help chapter PLC Automation with V3 CPUs > Diagnosis and debugging for AC500 V3 products > The diagnosis system > Diagnosis in IEC application for up-to-date information.

11.3 EtherCAT Diagnosis

Below chapter give an overview on the AC500 V3 EtherCAT diagnosis. For updated detailed information please refer to [AC500 EtherCAT diagnosis](#)

11.3.1 Application scenarios of EtherCAT diagnostics

In this chapter, typical faults and errors are listed which can be determined, detected and located by the EtherCAT bus.

The errors and faults are classified by Commissioning and Operational of the machinery. The list is based on practical experience with EtherCAT and consists predefined faults and errors derived from the ETG specification



Note: This document lists only a small selection of errors that can occur in an Ether-CAT system. It is only intended to provide an exemplary overview of the various error groups. Depending on the application and installation, further errors and error groups may occur. PM5012-x-ETH and all the PM50xx-R-ETH eCo PLC are not supported by motion solution wizard.

11.3.1.1 Commissioning

Especially during commissioning, the topology errors can occur frequently. With completion of the hardware setup, the topology errors are minimized and communication errors can emerge. Of course, some topology errors might be also caused by communication error, e.g. a loose connection of an EtherCAT device which leads to an incomplete system.

Topology error

During the transition from Bus Off to Init, i.e. during the start-up of the EtherCAT bus, the topology of the system will be checked by the master according to the configuration. The correct topology is mandatory for the proper operation of the bus. Normally, a topology error only occurs during commissioning.

In case of a specific application where single devices must shut down and replaced, these errors can occur as well.

Error or fault	Description
Additional devices	More devices installed than configured.
Missing devices	Less devices installed than configured.
Missing cable between two de-vices	This is basically the same behavior as the "Missing devices" error. All devices behind the missing cable are physically not available and therefore there are less devices installed than configured.
Reversed devices	Different types of EtherCAT Slave module, e.g. e190 servo drive instead of a e180 servo drive.
Reversed devices within one cluster	Salve modules are swapped.
Reversed devices of same type but with pre-defined station address	Module with predefined station address 1001 is swapped with another module with the address of 1002.
Reversed connection at In/Out Ports	During commissioning a reversed connection of the IN and OUT ports will cause that this device will be set as the last slave of the topology due to the different telegram processing inside the slave

11.3.1.2 Communication error

A communication error is an error when writing or reading the telegram and can therefore occurs at startup or operation. The correct operation of the communication is decisive for a functioning bus. In case of a communication error, further errors can be caused.

Error and fault	Description
Telegram error	E.g. Defect cable or port, EMC interferences.
Loose connection	Loose connection of EtherCAT cable or supply voltage.
Inadmissibly long or non-deter-ministic forwarding times	Faulty EtherCAT Slave Controller implementation or non-EtherCAT switch
Faulty device/slave	Loosing telegrams (e.g. telegrams will be rejected due to a broken line inside the slave) or telegram are not executed correctly.
Non-EtherCAT device	Loosing telegrams (The telegrams will not be returned, and therefore the telegrams are rejected).

11.3.2 Operational

Usually, topology errors do not occur during a running system unless the application requires a hardware change during operation, or a person willfully changes the topology.

Accordingly, only a communication error or a device, module or channel error can lead to a necessary diagnostic during operation.

11.3.2.1 Communication error

A communication error is an error when writing or reading the telegram and can therefore occur at startup or operation. The correct operation of the communication is decisive for a functioning bus. In case of a communication error, further errors can be caused.

Error and fault	Description
Telegram error	E.g. Defect cable or port, EMC interferences.
Loose connection	Loose connection of EtherCAT cable or supply voltage.
Inadmissibly long or non-deterministic forwarding times	Faulty EtherCAT Slave Controller implementation.
Faulty device/slave	Loosing telegrams (e.g. telegrams will be rejected due to a broken line inside the slave) or telegram are not executed correctly.
Non-EtherCAT device	Loosing telegrams (The telegrams will not be returned, and therefore the telegrams are rejected).

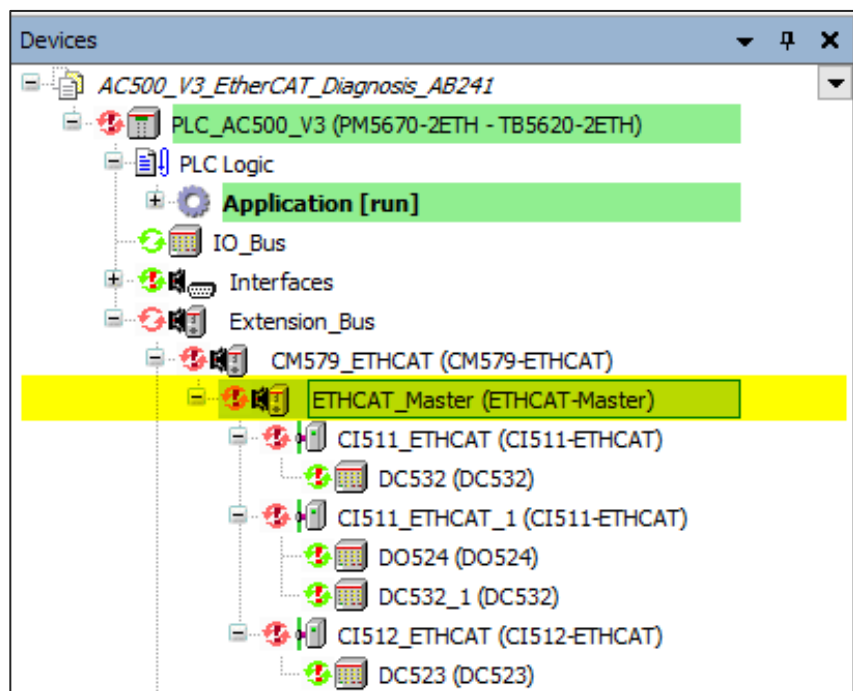
11.3.3 Diagnostic with Automation Builder

11.3.3.1 Diagnostic tools

Since the release of Automation Builder 2.2.4 (631), the Engineering Tool for the AC500, includes interfaces to read basic diagnosis information for commissioning purposes without any application effort.

This diagnostic has been improved over different Automation Builder Versions and is now completely implemented. This Guideline will explain the diagnostic interfaces since Automation Builder 2.3.0 and above.

Independent of the PLC type, the diagnosis can be found at the tab ETHCAT_Master (ETHCAT-Master) and is only available in Online mode.

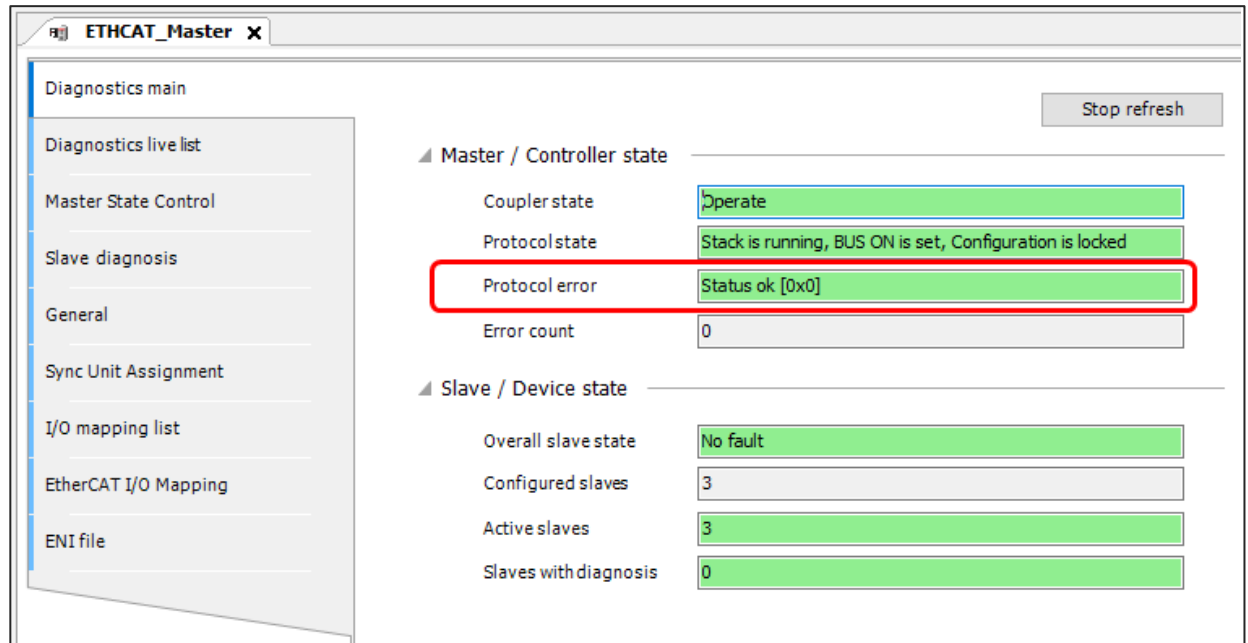


The following chapter will give an overview about the different diagnosis tools in Automation Builder.

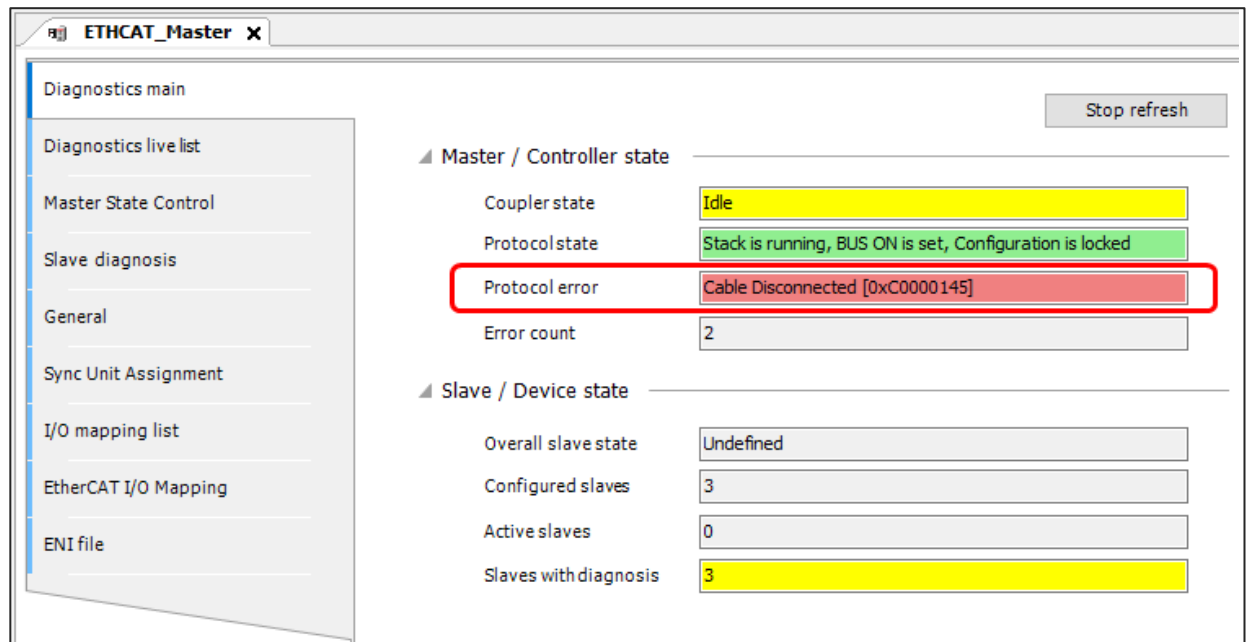
11.3.3.2 Diagnostics main

As general overview of the System, the Diagnostics main tab can be used. Accordingly, to the EtherCAT state machine (see also Chapter 2.3) the master and all slaves have to pass through all states for an operational bus. In the Diagnostics main tab, the Master and the Slave states are displayed and can be used to determine an error but not to locate the faulty device.

If the system goes to operational (OP), without having any error at the Master or the Slave, all display indications must be green.



In case of an error, the display indications change their color and the communication fault can be seen inside the Protocol error tab.



Having a look to the Slave/ Device state, it is noticeable that the number of configured and active slaves does not match each other. It is obvious that the connection from the master to the first slave already has an error.

11.3.3.3 Diagnostics live list

In case of a topology error the Diagnostics live list will list all found devices in their actual topology after pressing “Scan”. Beside the name, it will also display the state of the ports and the revision number of the devices.

ID	Type	State	Name	Address	Details
1	CI511-ETHCAT	Device is connected at Port 0 Device is connected at Port 1 DL Status: Link on Port 0 DL Status: Link on Port 1 DL Status: Communication on Port 0 DL Status: Communication on Port 1			Revision number: 0x00020102
2	CI511-ETHCAT	Device is connected at Port 0 Device is connected at Port 1 DL Status: Link on Port 0 DL Status: Link on Port 1 DL Status: Communication on Port 0 DL Status: Communication on Port 1			Revision number: 0x00020102
3	CI512-ETHCAT (device index C0 and above)	Device is connected at Port 0 DL Status: Link on Port 0 DL Status: Loop closed on Port 1 DL Status: Communication on Port 0			Revision number: 0x00020005

By comparing the scanned list with the configuration in the device tree, the incorrectly installed device and thus the topology error can be detected.

CAUTION! - Bus and Topology Scan will reset the System. As this is not a common function during operational and can causes unexpected issues, the scan is al-owed at the Init state only.

11.3.3.4 Master State Control

The state machine of the EtherCAT bus can be changed by the Master State Control register. Within this state control, the Master as well as the Slaves can be set to a desired state to start, initialize, or stop the entire system.

TimeStamp	Type	Message	Sender
2000-01-01 00:00:13.588	New master state	New state: INIT reached,	EtherCAT master
2000-01-01 00:00:13.993	New master state	New state: PREOP reached,	EtherCAT master
2000-01-01 00:00:19.988	New master state	New state: SAFEOP reached,	EtherCAT master
2000-01-01 00:00:20.008	New master state	New state: OP reached,	EtherCAT master
2000-01-01 00:02:29.504	Slave error	Station address: 1002, Type: 3, Param: C0CD0020,	1002
2000-01-01 00:02:29.505	Slave error	Station address: 1003, Type: 3, Param: C0CD0020,	1003
2000-01-01 00:02:29.549	Slave error	Station address: 1001, Type: 3, Param: C0CD0020,	1001

Next to the control pane for the EtherCAT system, the Master State Control register has additional windows to display diagnostic information:

1. Communication log:

The latest EtherCAT communication information is displayed on top to indicate the status of the EtherCAT network. In total the last five communication information of the master is shown.

2. Frame loss counters:

In case of lost frames, the counter will increase and therefore indicates whether some telegrams are lost in the network. The lost frame counter might increase during startup, therefore it must be read out for each startup and compared with the value during the fault.

3. Timing:

- Bus cycle time - Bus cycle time in nanoseconds
- Expected RX end time - Time from start of bus cycle transmission until completion of receiving the bus cycle back (in nanoseconds)
- Expected TX data time - Time from start of bus cycle transmission until a new data update is expected to be signaled to stack (in nanoseconds)

4. Log entry:

The existing Log file does not only show state changes of the bus, but also slave error with the hexadecimal code and the station address of the corresponding device. The additional slave errors can give deeper information about its faults (e.g. SDO error during startup of the system).

11.3.3.5 Slave diagnosis

New in Automation Builder 2.3 is the Slave diagnosis for the EtherCAT system. While the communication log of the Master State Control tab indicates whether an error is present or not, the Slave diagnosis can give additional information about the slaves directly and therefore about the positioning of the fault.

ETHCAT_Master x										
Stop refresh										
Diagnostics main										
Diagnostics live list										
Master State Control										
Slave diagnosis										
General										
Sync Unit Assignment										
I/O mapping list										
EtherCAT I/O Mapping										
ENI file										
Topology Position	Configured Station Address	Slave Name	Slave State	Port State	Last Error	Emergency	Frame Error Counters (Port 0-3)	Physical Layer Error Counters (Port 0-3)	Link Lost Counters (Port 0-3)	
1	0x1001	CIS11_ETHCAT	OP	Device is connected at Port 0 Device is connected at Port 1 DL Status: Link on Port 0 DL Status: Link on Port 1 DL Status: Communication on Port 0 DL Status: Communication on Port 1 DL Status: Loop closed on Port 2 DL Status: Loop closed on Port 3	Status ok [0x0]		0-0-0-0	0-0-0-0	0-0-0-0	
2	0x1002	CIS11_ETHCAT_1	OP	Device is connected at Port 0 Device is connected at Port 1 DL Status: Link on Port 0 DL Status: Link on Port 1 DL Status: Communication on Port 0 DL Status: Communication on Port 1 DL Status: Loop closed on Port 2 DL Status: Loop closed on Port 3	Status ok [0x0]		0-0-0-0	0-0-0-0	0-0-0-0	
3	0x1003	CIS12_ETHCAT	NOT CONNECTED	LLD: Timeout [0xC0CC0001]	Last connection [0xC0CD0020]		LLD: Timeout [0xC0CC0001]	LLD: Timeout [0xC0CC0001]	LLD: Timeout [0xC0CC0001]	

11.3.4 Process guideline for typical faults and errors during commissioning

11.3.4.1 Topology error

Error	Register	Description/ Explanation
Reversed devices		
Detection	Diagnostics main	<ul style="list-style-type: none"> Coupler state: Idle (\neq Operate) Protocol error: Topology mismatch detected [0xC0CD0044]
Localization	Diagnostics live list	Scan the bus and compare Types to find the faulty hardware position
Reversed devices within a cluster		
Detection	Diagnostics main/ Slave diagnosis	Protocol error/ Last error: AL Control Timeout happened i.e. a slave ESM state change was not completed in time [0xC0CD006B]
Localization	Slave diagnosis	Find topology position of faulty device by checking state <ul style="list-style-type: none"> Manually comparison of cluster configuration
	Master state control	Check Log entries – Stations address with corresponding error message will localize the faulty slave. <ul style="list-style-type: none"> Manually comparison of cluster configuration
Reversed devices of same type but with predefined station address		
Detection	Diagnostics main/ Slave diagnosis	Protocol error/ Last error: Device identification via register failed [0xC0CD008E]
Localization	Slave diagnosis	Last error displays "Device identification via register failed" and state remains in INIT + ERROR. <ul style="list-style-type: none"> Manually comparison of identification wheels
	Master state control	Check Log entries – Stations address with corresponding error message will localize the faulty slave. <ul style="list-style-type: none"> Manually comparison of identification wheels
Reversed connection at IN/OUT Ports		
Detection	Diagnostics main/ Slave diagnosis	Protocol error/ Last error: Topology error detected [0xC0CD0043]
Localization	Diagnostics live list	<p>Scan the bus and compare Types to find the faulty hardware position.</p> <p>Check the List of devices with the configuration. A reversed port causes that the slave will be set to the end of the bus assumed that only one reversed port is available.</p>
Missing cable between two devices		
Detection	Diagnostics main/ Slave diagnosis	Protocol error/ Last error: Topology error detected [0xC0CD0043]
Localization	Slave diagnosis	<p>Localization by topology position - Last error: Missing slave at port 1. [0xC0CD004C]</p> <p>Lost connection to the next slave.</p>
	Diagnostics live list	Result of the scan does not match the correct topology. All slaves after the missing cables are not listed.

Additional devices		
Detection	Diagnostics main	<ul style="list-style-type: none"> Coupler state: Idle (\neq Operate) Protocol error: Topology mismatch detected [0xC0CD0044]
Localization	Slave diagnosis	<p>Slave diagnosis will only be displayed at configured devices.</p> <ul style="list-style-type: none"> Last Error at last configured device: Unexpected slave at port 1 of slave. [0xC0CD0047]
	Diagnostics live list	<p>Result of the scan does not match the correct topology. The result must be compared manually to the configuration one.</p>
Missing devices		
Detection	Diagnostics main/ Slave diagnosis	Protocol error/ Last error: Topology error detected [0xC0CD0043]
Localization	Slave diagnosis	<p>Localization by topology position - Last error:</p> <p>Missing slave at port 1 of slave. [0xC0CD004C]</p> <p>Lost connection to the next slave.</p>
Telegram error		
Detection	Diagnostics main	<ul style="list-style-type: none"> Current state at Master Control \neq OP Protocol error: Topology error detected [0xC0CD0043] <p>A telegram error can affect the system in various ways. The protocol error "Topology error detected" is just an example on how the system might react to this kind of error.</p>
	Master State Control	<ul style="list-style-type: none"> Frame loss counters \neq 0 Message Log – Type, Index, Sub-index, Result
Localization	Slave diagnosis	<p>Localization by topology position of Slave – Error counter:</p> <ul style="list-style-type: none"> Frame Error Counter \neq 0 Physical Error Counter \neq 0 <p>Depending on the increasing error counters of the beside slaves, the root cause might be the cable or device itself.</p>
Loose contact		
Detection	Diagnostics main	<ul style="list-style-type: none"> CurrentState: PREOP (Bus state) or PreOpErr (Slave State) CommErno/ LastErr: Topology error detected [0xC0CD0043] <p>A loose contact can affect the system in various ways. The protocol error "Topology error detected" is just an example on how the system might react to this kind of error.</p>
	Master State Control	<ul style="list-style-type: none"> Frame loss counters \neq 0 Counter is increasing sporadically
Localization	Slave diagnosis	<p>Localization by topology position of Slave – Error counter:</p> <ul style="list-style-type: none"> Frame Error Counter \neq 0 Physical Error Counter \neq 0

11.3.4.2 Communication error

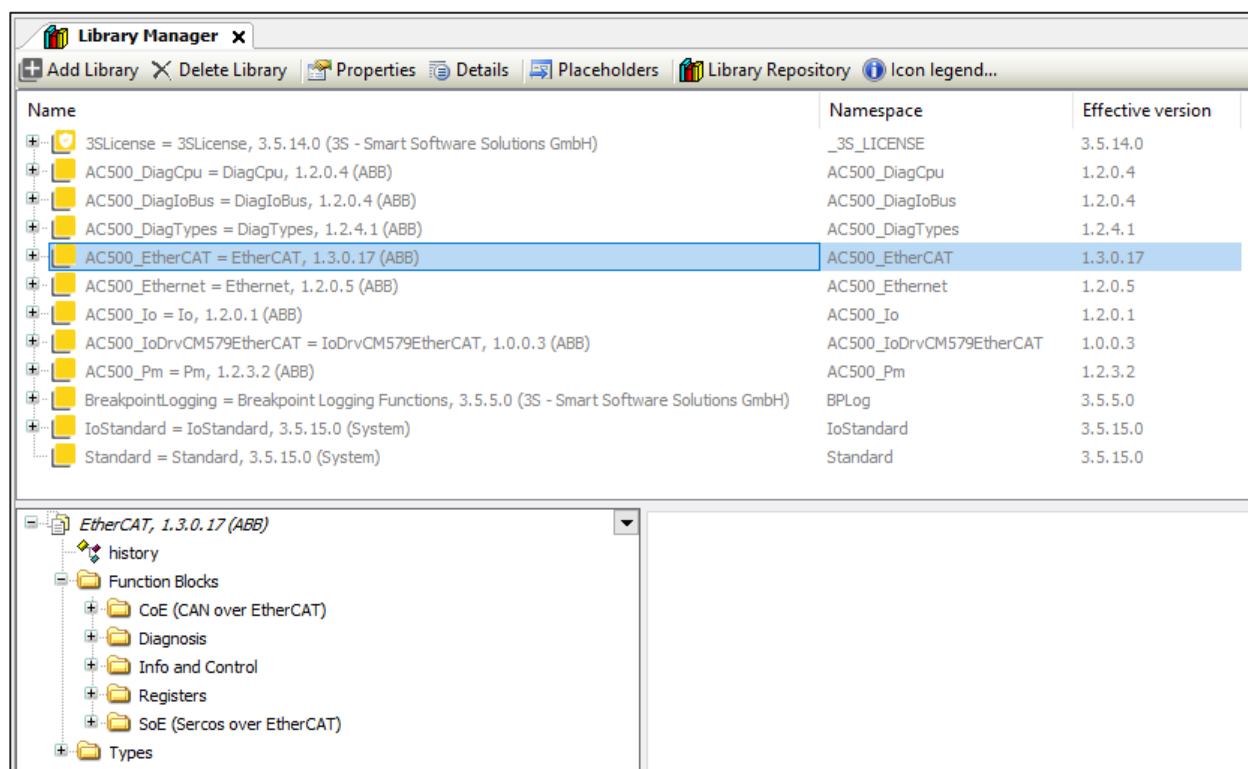
Error	Function block	Description/ Explanation
Telegram error		
Detection	Diagnostics main	<ul style="list-style-type: none"> Current state at Master Control = PREOP Protocol error: SDO Abort Code: Protocol Timeout [0xC0CF8002] <p>A telegram error can affect the system in various ways. The protocol error "SDO Abort Code" is just an example on how the system reacts to this kind of error.</p>
	Master State Control	<ul style="list-style-type: none"> Frame loss counters ≠ 0 Message Log – Type, Index, Sub-index, Result
Localization	Slave diagnosis	<p>Localization by topology position of Slave – Error counter:</p> <ul style="list-style-type: none"> Frame Error Counter ≠ 0 Physical Error Counter ≠ 0 <p>Depending on the increasing error counters of the be-side slaves, the root cause might be the cable or device itself.</p>
Loose contact		
Detection	Diagnostics main	<ul style="list-style-type: none"> CurrentState: PREOP (Bus state) or PreOpErr (Slave State) CommErno/ LastErr: SDO protocol timeout [0xC0CF8002] <p>A loose contact can affect the system in various ways. The protocol error "SDO protocol timeout" is just an example on how the system reacts to this kind of error.</p>
	Master State Control	<ul style="list-style-type: none"> Frame loss counters ≠ 0 Counter is increasing sporadically
Localization	Slave diagnosis	<p>Localization by topology position of Slave – Error counter:</p> <ul style="list-style-type: none"> Frame Error Counter ≠ 0 Physical Error Counter ≠ 0 <p>Depending on the increasing error counters of the be-side slaves, the root cause might be the cable or device itself.</p>
Inadmissibly long or non-deterministic forwarding times		
Detection	Diagnostics main/ Slave diagnosis	CommErno/ LastErr: DC RX TimeStamp Error [0xC0CD0026]
Localization	Slave diagnosis	Localization by topology position of Slave - LastErr: DC RX TimeStamp Error [0xC0CD0026]
Faulty device/slave		
Detection	Diagnostics main	<ul style="list-style-type: none"> Coupler state: Idle Protocol error: Cable Disconnected [0xC0000145]
	Master State Control	<ul style="list-style-type: none"> Frame loss counters ≠ 0

11.4 Diagnostic with IEC programming

Automation Builder serves libraries with function blocks for extended EtherCAT diagnostic. System integrators have access to the whole diagnostic System of EtherCAT and can implement it individually to provide diagnostic for different purposes. Therefore, diagnostic during commissioning is not only available in Automation Builder but also in IEC code.

To implement EtherCAT diagnostic inside IEC code for a V3 PLC, the Library AC500_EtherCAT is required. The library will be added to the Library Manager after adding the CM579-ETHCAT to a PLC slot. At least Version 1.3.0.17 of the Library is required to use Automation Builder 2.3.0 and a PM5670-ETH PLC with the Firmware Version 3.3.1.0.

The Library consists of different folders with Function blocks to read/ write CoE- and SoE data or to read/ write registers, to control the EtherCAT system and to get diagnostic of the same



This chapter shows the process how to detect and localize faults and errors that might occurs during commissioning by using the library in the IEC code.

11.4.1 Topology error

Error	Function block	Description/ Explanation
Reversed devices		
Detection	EcatBusDiag	CurrentState: INIT (≠ Operate) CommErno: Topology mismatch detected [0xC0CD0044]
Localization	EcatScanTopology & EcatSlvDiag	<ul style="list-style-type: none">Scan the bus and compare Types to find the faulty hardware positionLastErr: Wrong slave at position [0xC0CD0035]
Reversed devices within a cluster		
Detection	EcatBusDiag/ EcatSlvDiag	CommErno/ LastErr: AL Control Timeout [0xC0CD006B]
Localization		DiagData: Wrong type or model.
Reversed devices of same type but with predefined station address		
Detection	EcatBusDiag/ EcatSlvDiag	Protocol error/ Last error: Explicit device identification failed (register) [0xC0CD008E]
Localization		Find topology position of faulty device by checking state <ul style="list-style-type: none">Manually comparison of cluster configuration
Reversed connection at IN/OUT Ports		
Detection	EcatBusDiag	CommErno: Topology mismatch detected. [0xC0CD0044]
Localization	EcatScanTopology (CheckTopology = True)	<ul style="list-style-type: none">Check the List of Devices with the configuration. A reversed port causes that the slave will be set to the end of the bus assumed that only one reversed port is available.Input "CheckTopology" is available at the EcatScan-Topology function block. If this input is set to True, the function block is checking the bus for reversed ports. → ReversedPortsPosition
Missing cable between two devices		
Detection	EcatBusDiag/ EcatSlvDiag	CommErno/ LastErr: Topology mismatch detected [0xC0CD0044]
Localization	EcatSlvDiag	Localization by topology position - LastErr: Missing slave at port 1. [0xC0CD004C] Lost connection to the next slave.
Additional devices		
Detection	EcatBusDiag	CommErno: Topology mismatch detected. [0xC0CD0044]
Localization	EcatSlvDiag	Slave diagnosis will only be displayed at configured devices. LastErr at last configured device: Unexpected slave at port 1 of slave. [0xC0CD0047]

	EcatScanTopology	Result of the scan does not match the correct topology. The result must be compared manually to the configuration one.
Missing devices		
Detection	EcatBusDiag/ EcatSlvDiag	CommErno/ LastErr: Topology mismatch detected [0xC0CD0044]
Localization	EcatSlvDiag	Localization by topology position - LastErr: Missing slave at port 1. [0xC0CD004C] Lost connection to the next slave.
Telegram error		
Detection	EcatBusDiag	<ul style="list-style-type: none"> • Current state at Master Control \neq OP • Protocol error: Topology error detected [0xC0CD0043] <p>A telegram error can affect the system in various ways. The protocol error "Topology error detected" is just an example on how the system might react to this kind of error.</p>
	EcatMasterGetTresholdCnt or EcatMasterGet-FrameLossCount	<ul style="list-style-type: none"> • Frame loss counters \neq 0 • Message Log – Type, Index, Sub-index, Result
Localization	EcatSlvDiag	<p>Localization by topology position of Slave – Error counter:</p> <ul style="list-style-type: none"> • Frame Error Counter \neq 0 • Physical Error Counter \neq 0 • Depending on the increasing error counters of the beside slaves, the root cause might be the cable or device itself.
Loose contact		
Detection	EcatBusDiag/ EcatSlvDiag	<ul style="list-style-type: none"> • CurrentState: PREOP (Bus state) or PreOpErr (Slave State) • CommErno/ LastErr: Topology error detected [0xC0CD0043] <p>A loose contact can affect the system in various ways. The protocol error "Topology error detected" is just an example on how the system might react to this kind of error.</p>
	EcatMasterGetTresholdCnt or EcatMasterGet-FrameLossCount	<ul style="list-style-type: none"> • Frame loss counters \neq 0 • Counter is increasing sporadically
Localization	EcatSlvDiag	<p>Localization by topology position of Slave – Error counter:</p> <ul style="list-style-type: none"> • Frame Error Counter \neq 0 • Physical Error Counter \neq 0 <p>Depending on the increasing error counters of the beside slaves, the root cause might be the cable or device itself.</p>

11.4.2 Communication error

Error	Function block	Description/ Explanation
Telegram error		
Detection	EcatBusDiag/ EcatSlaveDiag	<ul style="list-style-type: none"> CurrentState: PREOP (Bus state) or PreOpErr (Slave State) CommErrno/ LastErr: SDO protocol timeout [0xC0CF8002] <p>A telegram error can affect the system in various ways. The protocol error "SDO protocol timeout" is just an example on how the system might react to this kind of error.</p>
	EcatMasterGetTresholdCnt or EcatMasterGet-FrameLossCount	<ul style="list-style-type: none"> TresholdCounters/ LostFrames ≠ 0 Counter is increasing sporadically
Localization	EcatSlvReadRxErrorCnt	<p>Localization by topology position of Slave – Error counter:</p> <ul style="list-style-type: none"> Frame Error Counter ≠ 0 Physical Error Counter ≠ 0 <p>Depending on the increasing error counters of the beside slaves, the root cause might be the cable or device itself.</p>
Loose contact		
Detection	EcatBusDiag/ EcatSlaveDiag	<ul style="list-style-type: none"> CurrentState: PREOP (Bus state) or PreOpErr (Slave State) CommErrno/ LastErr: SDO protocol timeout [0xC0CF8002] <p>A telegram error can affect the system in various ways. The protocol error "SDO protocol timeout" is just an example on how the system might react to this kind of error.</p>
	EcatMasterGetTresholdCnt or EcatMasterGet-FrameLossCount	<ul style="list-style-type: none"> TresholdCounters/ LostFrames ≠ 0 Counter is increasing sporadically
Localization	EcatSlvReadRxErrorCnt/ EcatSlavReadLostLinkCnt (optional)	<p>Localization by topology position of Slave – Error counter:</p> <ul style="list-style-type: none"> Frame Error Counter ≠ 0 Physical Error Counter ≠ 0 Lost Link Error Counter ≠ 0 <p>Depending on the increasing error counters of the beside slaves, the root cause might be the cable or device itself.</p>
Inadmissibly long or non-deterministic forwarding times		
Detection	EcatBusDiag/ EcatSlaveDiag	CommErrno/ LastErr: DC RX TimeStamp Error [0xC0CD0026]
Localization	EcatSlaveDiag	Localization by topology position of Slave – LastErr: DC RX TimeStamp Error [0xC0CD0026]
Faulty device/slave		
Detection	EcatBusDiag	<ul style="list-style-type: none"> CurrentState: INIT (≠ Operate) CommErrno: Cable Disconnected [0xC0000145]

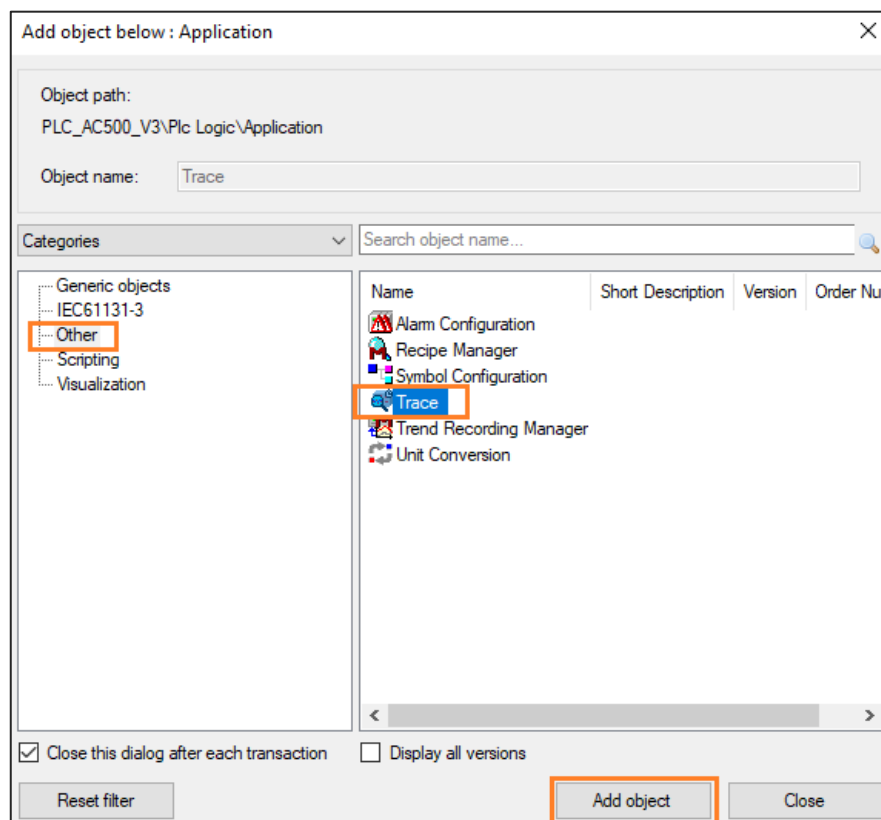
	EcatMasterGetTresholdCnt or EcatMasterGet- FrameLossCount	<ul style="list-style-type: none"> • TresholdCounters/ LostFrames ≠ 0
	Special case: It might happen that a faulty slave does not executes telegrams correctly which affects that the Working counter is invalid while the TresholdCounters/ LostFrames remains 0 .	
Localization	EcatEmergencyScan	The emergency scan lists the proper working devices according to the topology. Therefore, the device at the end of the list, indicates the last working slave. The device after is the non EtherCAT device.
Non-EtherCAT device		
Detection	EcatBusDiag	<ul style="list-style-type: none"> • CurrentState: INIT (≠ Operate) • CommErno: Cable Disconnected [0xC0000145]
	EcatMasterGetTresholdCnt or EcatMasterGet- FrameLossCount	<ul style="list-style-type: none"> • TresholdCounters/ LostFrames ≠ 0
Localization	EcatEmergencyScan	The emergency scan lists the proper working devices according to the topology. Therefore, the device at the end of the list, indicates the last working slave. The device after is the non EtherCAT device.

11.5 Data recording with trace

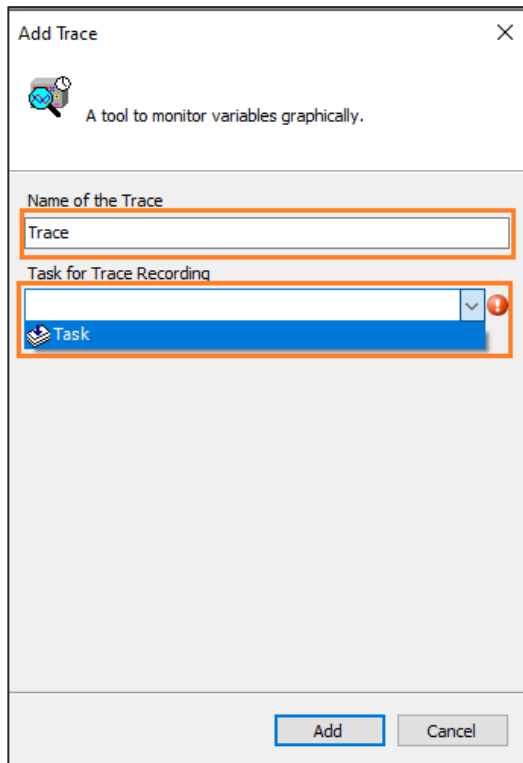
You can use a “Trace” to follow the value history of variables on the controller in a similar way as a digital sampling oscilloscope. To add the trace to Automation Builder device tree,

Right click on the “Application” and click on the Add Object.

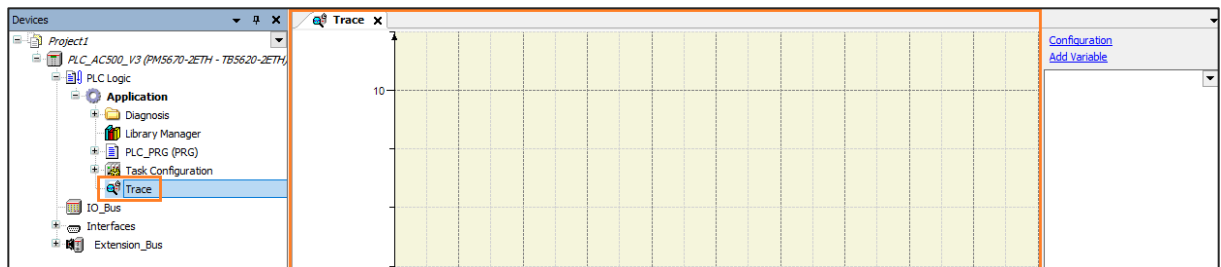
This will open the Add Object window, select “Other” and “Trace as shown below” and click on “Add object”.



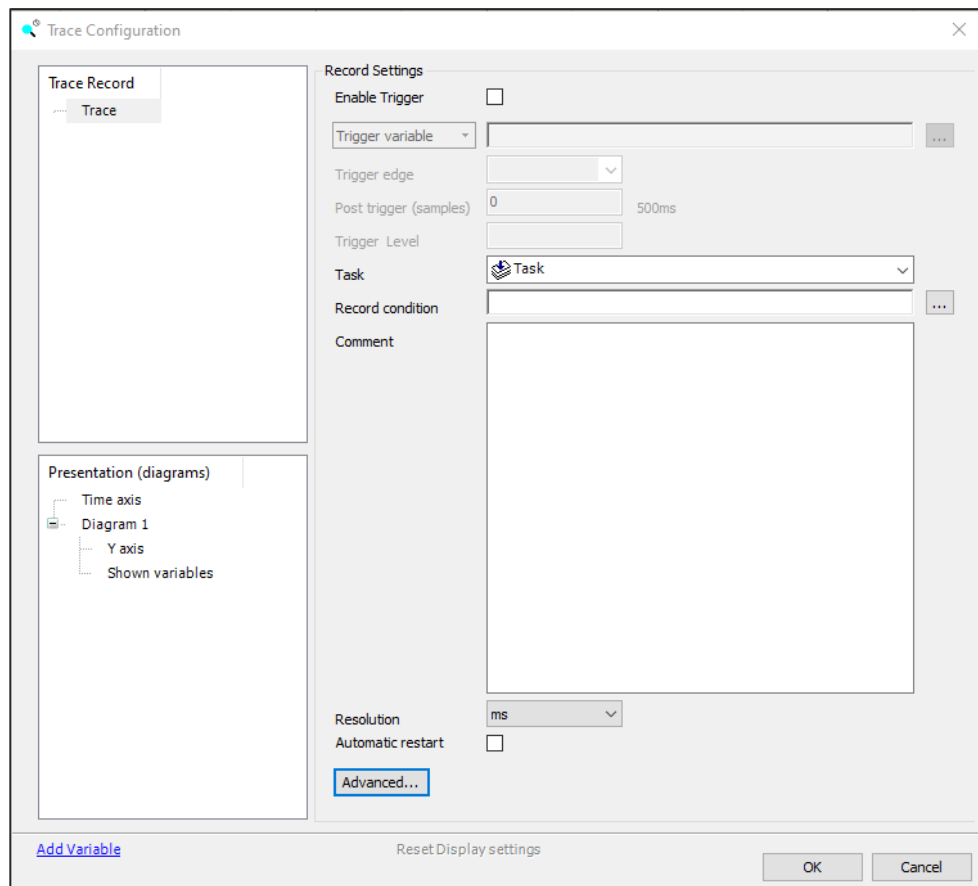
“Add Trace” window will open and provide a name and assign a task for the trace and click on “Add”.



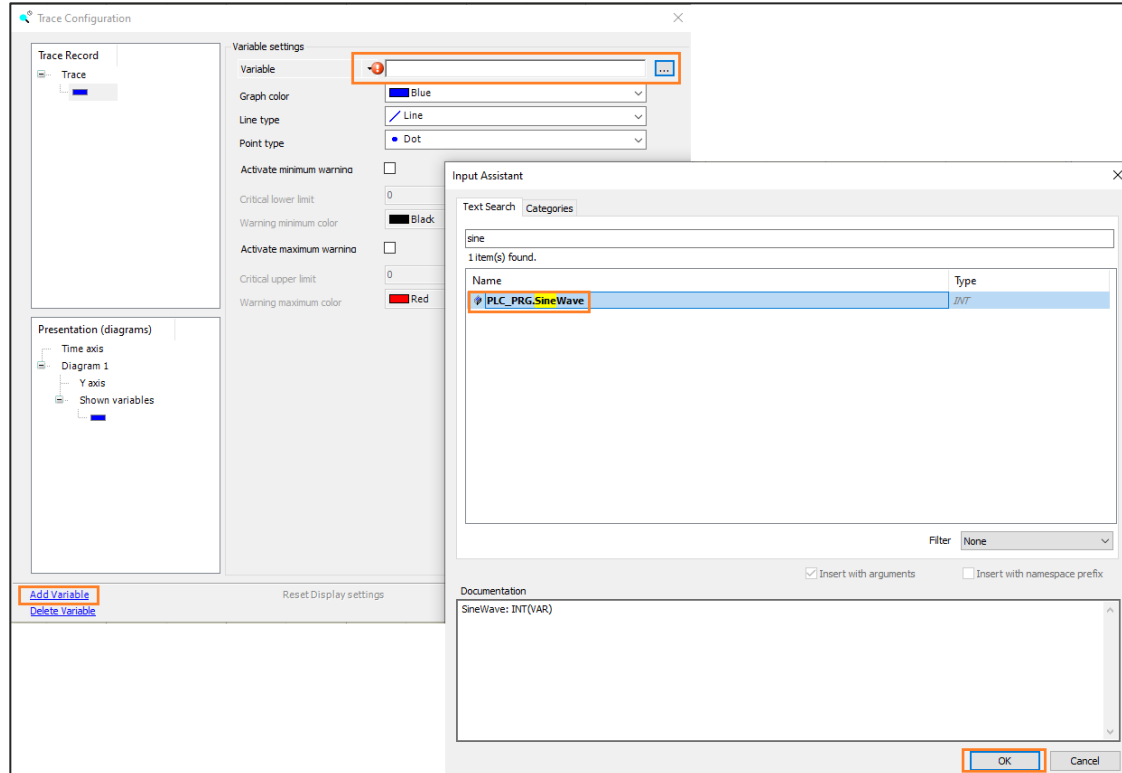
This will add the trace to device tree.



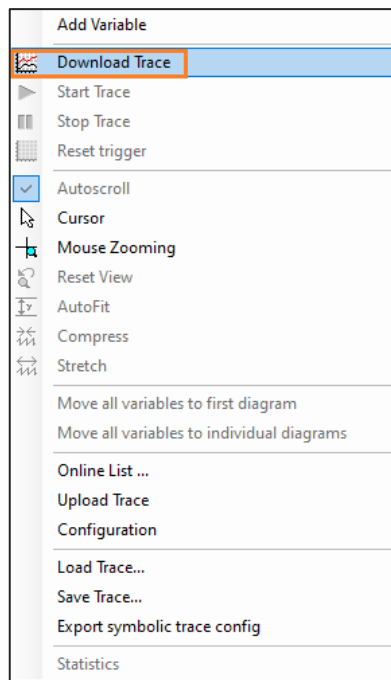
User can configure the trace by clicking on the “configuration”, which will the pop-up Trace configuration window. Details on the configuration can be found from latest Automation Builder help file.



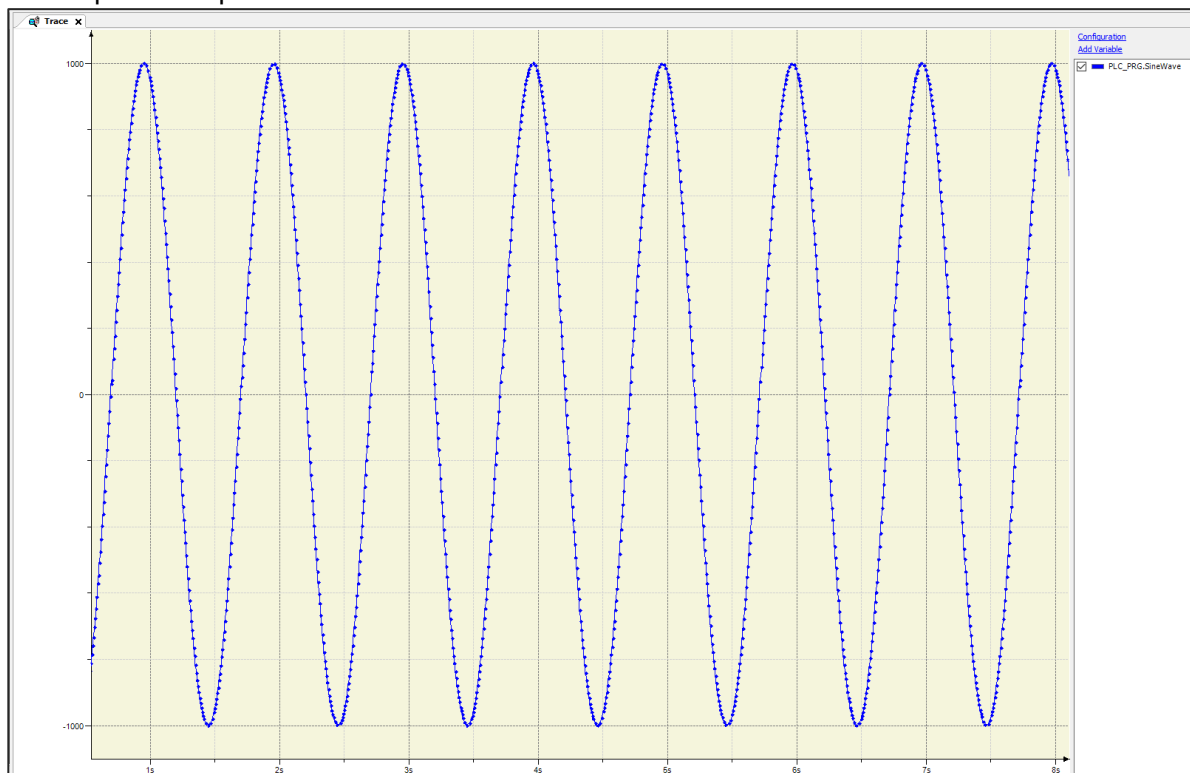
To add a variable to monitor, user can click on Add variable and map the variable.



After adding the variable to Trace, login to the PLC, right click on the trace area and click on "Download Trace".



Trace will be downloaded based on the configuration and user can right click on the trace to get more option to operate the trace.



REVISION HISTORY

Rev.	Page	Change Description	Date / Initial
-d1	All	First version	2022-03-10 MODP/AC500/DKO
-d2		Review and accounted for changes by Richard	2022-04-28 MODP/AC500/GS
-d3 1.2.4		Cleaning and accounting for review comments from August and before from CN Team, Additions of eCo in chapter2, Diagnosis chapter 12 and update of Configurator Chapters 8 and CAM chapter 9	2022-12-19 MODP/AC500/DKO, GS

D = draft

R = released version

ABB AG
Eppelheimer Straße 82
D-69123 Heidelberg / Germany
Phone: +49 62 21 701 1444
Fax : +49 62 21 701 1382
E-Mail: plc.support@de.abb.com

new.abb.com/plc
new.abb.com/plc/automationbuilder
new.abb.com/contact-centers

We reserve the right to make technical changes or modify the contents of this document without prior notice. With regard to purchase orders, the agreed particulars shall prevail. ABB AG does not accept any responsibility whatsoever for potential errors or possible lack of information in this document.

We reserve all rights in this document and in the subject matter and illustrations contained therein. Any reproduction, disclosure to third parties or utilization of its contents – in whole or in parts – is forbidden without prior written consent of ABB AG.
Copyright© 2022 ABB. All rights reserved