TM440

Motion Control: Basic Functions

PERFECTION IN AUTOMATION
www.br-automation.com

Prerequisites and requirements

Training modules	TM240 – Ladder Diagram (LD) or TM246 – Structured Text (ST) TM410 – Working with Integrated Motion Control
Software	Automation Studio 4.2 Automation Runtime 4.08 mapp Technology 1.00.0
Hardware	X20 controller + ACOPOS / ACOPOSmulti or Simulation

Table of contents

1 Introduction

The application program is a fundamental part of each positioning task.

This is where signals from the process are evaluated and movement commands are configured and transferred to the drive. The application program controls the drives used in the process, usually with an automatic sequence. The diagnostics for the drives also includes the application software and the visualization system.

The basis for designing a positioning application is to create an overview of the tools that are available.

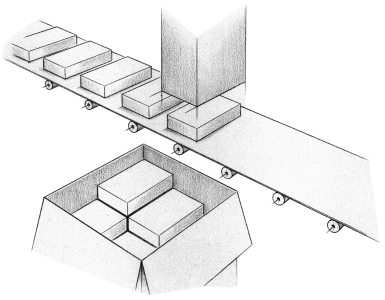


Figure 1: Schematic illustration of a palletizer

This training module deals with the use of mapp technology components from the MpAxis library and its integration into the application program.

Working through different exercises will help us to understand the behavior of function blocks and provide us with important basic knowledge for applying these functions. Ladder Diagram is used in the image source files and examples for calling function blocks. Any programming language can be used for creating the application program.

1.1 Learning objectives

With the aid of selected exercises, you get an overview of the application and integration of PLCopen-compliant standard functions. They are used for drive preparation and performing basic movements.

- You will get an overview of the mapp technology concept.
- You will know the structure of the MpAxis library and the contained function blocks.
- You will learn how to integrate PLCopen-compliant function blocks into your control project.
- You will get to know the necessary steps for drive preparation and performing basic movements.
- You will know the typical structure of an application program and understand the relationships in the PLCopen state diagram.
- You will understand what "axis period" and "axis factor" mean and will be able to calculate and configure them.
- You will know about configuration management for the MpAxis components and will be able to load and save the drive configuration.
- You will know the approach for incorporating status evaluation in the application program.
- You will have an overview of the ACP10_MC library structure.

1.2 PLCopen standard

The role played by software in automation systems is continuously increasing. This increases the complexity of applications, which results in additional work for development, support and maintenance of software.

The PLCopen organization was created with the goal of unifying and standardizing different areas of activity in the field of automation. The areas of activity include drive technology, safety technology, the IEC 61131-3 standard as well as OPC UA and XML.

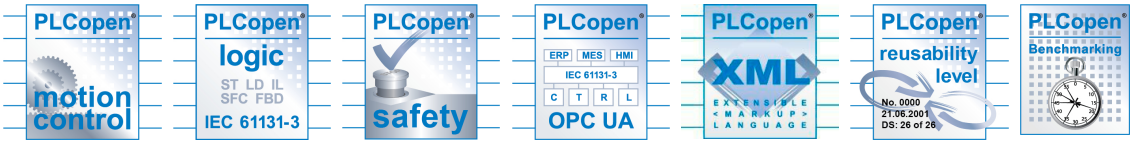


Table 1: Different fields of activity of the PLCopen organization

Detailed information about the PLCopen organization and its activities can be found on its website: **http://www.plcopen.org/**

Guidelines are being developed to ensure that different system solutions are all operated in the same way.

Suppliers of automation solutions who are members of this organization provide uniform software interfaces that are defined via PLCopen. B&R is an active member of PLCopen organization.

Fully supported by B&R

PLCopen-compliant function blocks are available for programming the B&R drive solution. Project creation is done quickly and easily using standardized function blocks.

2 mapp technology

2.1 Instructions for using mapp technology components

The following steps have to be carried out when using a mapp component for the first time.

- Go to the Configuration View
- Add the "mapp" technology package from the toolbox
- Add the standard configuration for the mapp component being used from the toolbox
- Rename the MpLink in the Configuration View as needed
  The Configuration View should look like this:

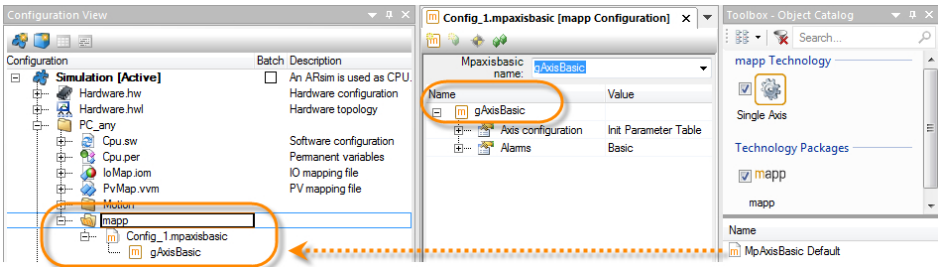


Figure 2: Representation of the Configuration View with mapp technology package and the standard configuration for the MpAxisBasic function.

The MpLink from the Configuration View is transferred to the function block in the program with the ADR() function. This establishes the connection from the configuration to the programming code of the mapp component.

Application layer - mapp technology \ Concept \ Component design \ Adding mapp components

Calling mapp components

The mapp component function blocks should be called in every controller cycle. When using high-level programming (Ansi C, ST, etc.), it is advisable to call all mapp components at the end of the program.

Every mapp function block has an "Enable" input. This input is used to enable the mapp component, which causes the configuration for the respective mapp component to be loaded automatically. Successful initialization of the mapp component is displayed on the "Active = TRUE" output.

Application layer - mapp technology \ Concept \ Component design \ Using mapp components

Download behavior

The controller has to be restarted using the standard settings in Automation Studio each time the program has been transferred. It is advisable to use the "Copy mode" transfer method during the implementation phase of the drive application. By doing so, it is no longer required to restart the controller a few of the times. The configuration for the type of transfer is opened using the shortcut menu of the active configuration in Configuration View. The transfer method is set in the Transfer tab using the Advanced button.

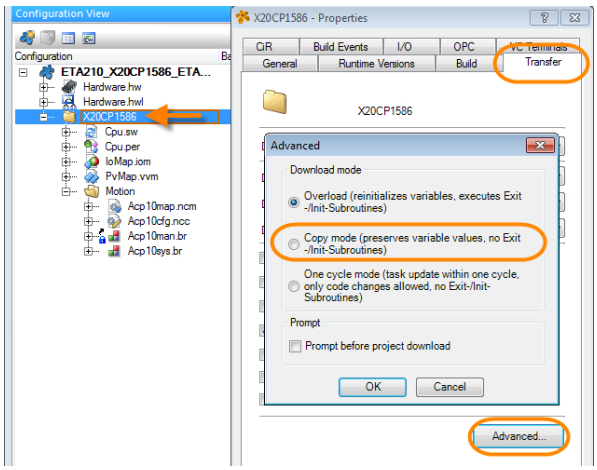


Figure 3: Configuration of the transfer method in the Properties window in Configuration View

Application layer - mapp technology \ Concept \ Component design \ Using mapp components

Real-time operating system \ Target systems \ SG4 \ Download

Configuration Files

There is a configuration available for every mapp component. The configuration is created and modified in the Configuration View in Automation Studio, the WebXs web-based interface or the application program. Additional information about mapp configuration can be found in the Automation Studio help system.

Application layer - mapp technology \ mapp \ Concept

- Component design \ Adding mapp components
- Configuring components

2.2 Diagnostic options for mapp technology components

mapp technology components can be monitored and diagnosed via several different methods. The following is a list of the diagnostic options in Automation Studio, in web-based diagnostics and in the visualization application.

Programming languages in monitor mode

In many cases, the first access is monitor mode during application software programming. The values of the process variables are visible in context directly with the program code. All mapp technology components have the "Error" and "StatusID" outputs that can be used to perform initial diagnostics.

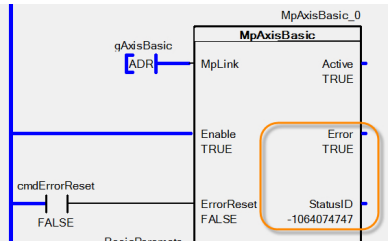


Figure 4: Ladder Diagram program in monitor mode

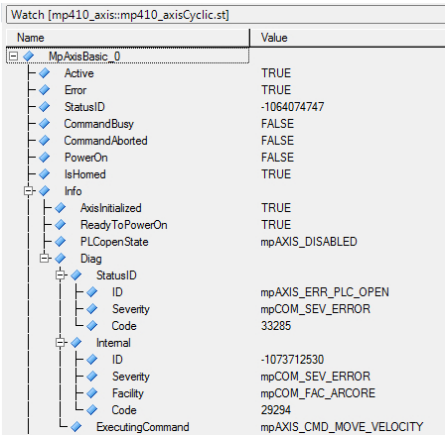


Figure 5: Instance variable of the MpAxisBasic function block in the Watch window

Watch window

The Watch window is opened in Logical View using the program shortcut menu or in the software configuration by selecting **<Watch>** . The instance variables of the function blocks used are added using the toolbar or shortcut menu. For example, the Error, StatusID, CommandBusy outputs and the information structure can help to diagnose the current state. A description of these parameters as well as error numbers is available to read in the description of the respective function block.

Logger

In the case of an error, additional information from the mapp technology component is added into the logger file with the name "$mapp". The error number can be searched for directly in the Automation Studio help system or called by pressing the **<F1> key** . Further information is available in the details section of the highlighted logger entry. For example, if there is a PLCopen error, the affected function and the cause of the error are described clearly in the details section of the logger entry.

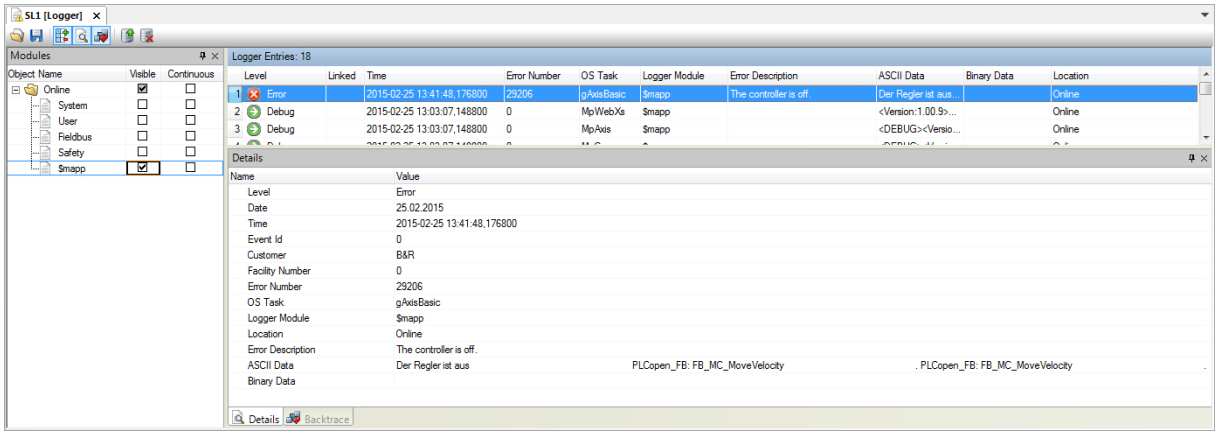


Figure 6: PLCopen error in the logger window

Trace

With the Automation Studio trace function, values of process variables are recorded in real-time and saved. Above all, the timing of input parameters and status variables can be visualized well.

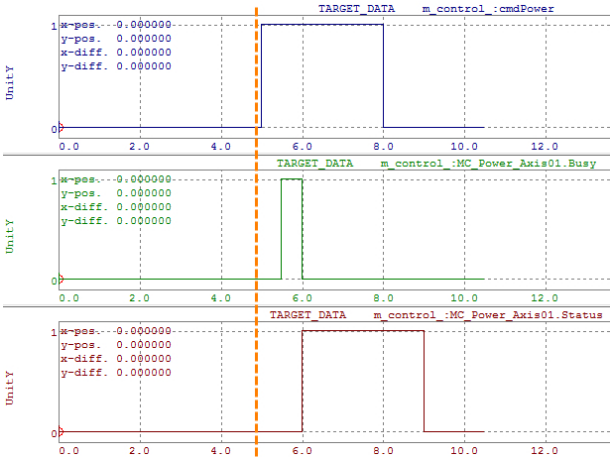


Figure 7: Switching on the controller with cmdPower: the time relationship between commands and status information

System Diagnostics Manager

The "Application Status" button can be called using the System Diagnostics Manager. This directly opens WebXs for mapp technology. Furthermore, saving the logger data for the mapp components in SDM is enabled. There are also basic diagnostics for drive axes available. The SDM can be embedded directly into a Visual Components application using the HTML control.

mapp WebXs

Using mapp technology WebXs, all mapp components being used are shown on a web-based interface. Configuration of the mapp components and alarms is also offered in addition to diagnostics via the instance variables for the components[1].

Integration of Visual Components using the MpAlarm component

The mapp technology components have predefined alarms. User-specific alarms can also be configured. The output of mapp alarms in the Visual Components alarm system is enabled using the MpAlarm component.

Integration of Visual Components using the MpComLoggerUI component

Event management features of mapp technology store all events in the logger. These logger entries can be easily integrated in Visual Components using the MpComLoggerUI component. Filter functions make it possible to search for individual mapp components, certain error numbers or event types. Additional programming is therefore not required to filter out mapp logger entries.

[1] Whether a web-based configuration can be carried out depends on the component used.

Diagnostics and service \ Diagnostic tool \

- Logger
- Watch window
- Monitors \ programming languages in monitor mode
- Trace
- System Diagnostics Manager

Application layer - mapp technology \

- WebXs
- Components \ Infrastructure \
  - MpAlarm - Support for alarm management
  - MpCom - mapp management \ function blocks \ MpComLoggerUI
- Diagnostics \ Logger window

3 Integrating an axis in the control project

The procedure for creating a drive configuration in Automation Studio has been explained in the training module "TM410 – Working with Integrated Motion Control". Drive movements could already be performed using NC Test.

The following example shows how an application program for controlling axis movements is created step by step.

An axis reference for accessing the axis object will be needed first. This axis reference has already been generated by the Drive Configuration wizard and added to the global variable declaration and the NC mapping table.

The drive is initialized automatically with the data from the NC Init module (which is assigned by the NC mapping table) when the controller is started up.

Motion \ Projection configuration \ Motion control

- Setting up an axis
- Configuration module \ NC mapping table
- Configuration module \ NC Init module

Getting started

The respective Getting Started section shows how the MpAxisBasic mapp technology component is inserted for performing drive preparation and basic movements.

Application layer - mapp technology \ Getting Started \ Quickly starting an axis

3.1 The MpAxis component

The MpAxis mapp technology component offers standard functions for controlling and configuring drive axes. The MpAxisBasic function block is used for drive control. The MpAxisBasicConfig function block is used for managing the drive configuration. The following function groups for single-axis control are covered by the MpAxisBasic function block:

- Drive preparation
- Basic movements
- Autotuning
- Error handling
- Cyclic drive data

The MpAxisBasic library is based on the function blocks in the ACP10_MC library. Both libraries are therefore compatible with one another and can be implemented together in applications (see 7.2 "Compatibility of ACP10_MC and MpAxis" on page 33).

All mapp components are based on open standards, technology functions and libraries that can be directly used by the user in the application.

Details regarding the used function blocks and functionality can be taken from the corresponding section in the Automation Studio help system.

Figure 8: mapp components are based on open standards, technology functions and libraries

Application layer - mapp technology \ Components \ Mechatronics \ MpAxis - individual and multi-axis controllers \ Technical information \ The use of internal PLCopen

Programming \ Libraries \ Motion libraries \ ACP10_MC

3.2 Creating a program and adding MpAxisBasic

Now it is necessary to expand the Automation Studio project by calling the MpAxisBasic function block. In a preparatory step, the project in the Configuration View is expanded by adding the mapp technology package. Next, the standard configuration for the MpAxisBasic component is added to the Configuration View. The MpAxis library is added to the Logical View.

At the end, the MpAxisBasic function block is added to a program and connected with the MpLink that was already set up in the Configuration View.

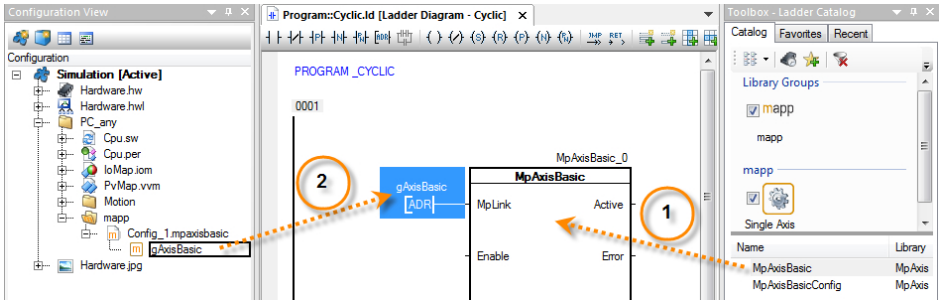


Figure 9: (1) Add MpAxisBasic from the toolbox and (2) transfer the MpLink address from the Configuration View

Application layer - mapp technology \ Concept \ Component design \ Adding mapp components

Task: Adding the mapp technology configuration for MpAxisBasic

Now expand the Automation Studio project by adding the mapp technology package and then insert the standard configuration for the MpAxisBasic component. Import the MpAxis library, next create a new program, add the MpAxisBasic function block and then connect the MpLink from the Configuration View using the ADR() function.

1) Go to the Configuration View
2) Add the mapp technology package from the toolbox
3) Add the MpAxisBasic standard configuration in the Configuration View
4) Add the MpAxis library in the Logical View
5) Add the new "m_control" Ladder diagram program
6) Add the MpAxisBasic function block from the toolbox
7) Assign MpLink from the Configuration View to the MpAxisBasic function block using the ADR() function

Now the project is prepared enough so that the desired drive can be accessed via the mapp component.

3.3 Connect the axis reference and the movement parameters

It is necessary to transfer the axis reference and the movement parameters so that an axis can be accessed and basic movements can be performed.

Using the axis reference

With the Drive Configuration wizard, a global process variable of type ACP10AXIS_typ is set up. The name of the process variable is automatically entered into the NC mapping table. The connection is then made from the axis object to the actual drive hardware via the POWERLINK network.

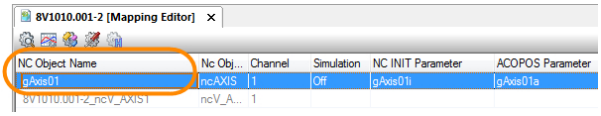


Figure 10: Axis reference in the NC mapping table

The address of the axis reference is specified for all function blocks using the "Axis" parameter.

In Ladder Diagram, either the address function (ADR) or an address contact can be used.

Access to the axis happens in the same way for all other function blocks.

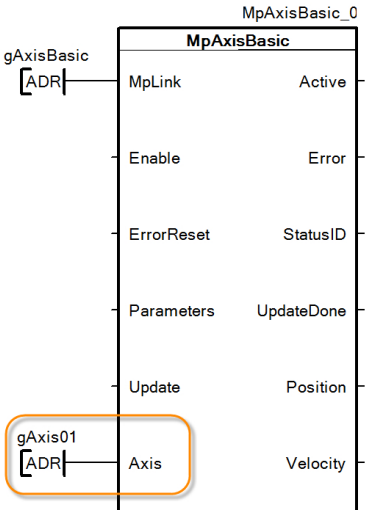


Figure 11: The axis reference is specified for the MpAxisBasic function block via the Address contact.

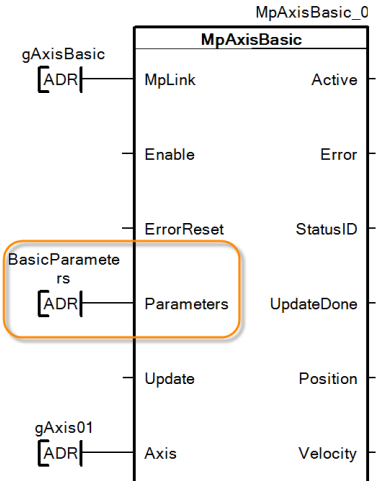


Figure 12: Transfer of the data structure with the movement parameters (MpAxisBasicParType)

Transferring movement parameters

For the MpAxisBasic function block, it is necessary to transfer a data structure with the movement parameters. The data structure is preinstalled with the standard values The following movement parameters can be transferred:

- Speed and acceleration
- Distance, position and direction of rotation
- Homing parameters
- Jog parameter
- Torque control
- Settings for the cyclic reading of axis information
- Autotuning

It should be taken into consideration that the initialization value "0" is automatically entered when setting up the data structures in the variable declaration. To allow default values to be used for the data structure, the initialization value must be deleted from the variable declaration.

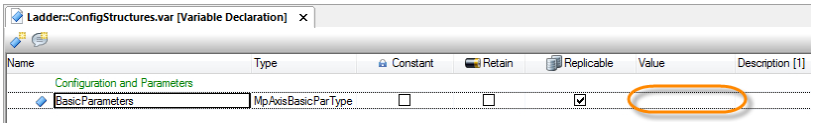


Figure 13: Delete the initialization values from the variable declaration to initialize with default values.

Application layer - mapp technology \ Components \ Mechatronics \ MpAxis - individual and multi-axis controllers \ Function blocks \ MpAxisBasic

Programming \ Libraries \ Motion libraries \ ACP10_MC \ Concept \ Implementation

Exercise: Assigning the axis reference and switching on the controller

The existing program is now expanded so that the axis reference and the data structure with the basic parameters can be transferred to the MpAxisBasic function block.

The "Enable" input must be set to TRUE to enable the MpAxisBasic component.

It is necessary to turn on the drive in order to prepare the drive[2]. If the "Active" output = TRUE, then the controller is switched on via the "Power" input of the MpAxisBasic function block.

1) Assign an axis reference to the input using the address function
2) Connect the "BasisParameters" structure with the "Parameters" input
3) Transfer the program to the controller
4) Set "Enable" input to TRUE
5) Wait until the "Active" output and the "Info.ReadyToPowerOn" output are TRUE.
6) Switch on the drive via "Power"
7) Observe the status outputs of MpAxisBasic

The "BasisParameters" structure is based on the MpAxisBasicParType data type. The structure is preinitialized with default values. Take into consideration that, for example, the configured acceleration and speed have to match the configured system of units.

Application layer - mapp technology \ Components \ Mechatronics \ MpAxis - individual and multi-axis controllers \ Function blocks \ MpAxisBasic

Programming \ Libraries \ Motion libraries \ ACP10_MC \ Function blocks \ Drive preparation

[2] With the ACOPOSmulti system, power supply modules should also be switched on via the software

3.4 Function block operation and status evaluation

This section briefly points out the operation of PLCopen-compliant function blocks. We will look at how to operate the function blocks as well as the options available for monitoring their operation. All function blocks are accessed using uniform operating parameters and return uniform status information. This simplifies the application and adds clarity during programming.

Timing diagrams

In the Automation Studio help system, how the input states of the function blocks operate is illustrated with the aid of timing diagrams.

The example shows that the "Power" function is only available if the component is active. It is necessary to set the "Enable" input to enable the component. If the "Enable" input is disabled when a controller is switched on and a movement is active, then the movement is immediately aborted and the controller is switched off. The controller is only switched on again after the component is reactivated and a rising edge occurs on the "Power" input.

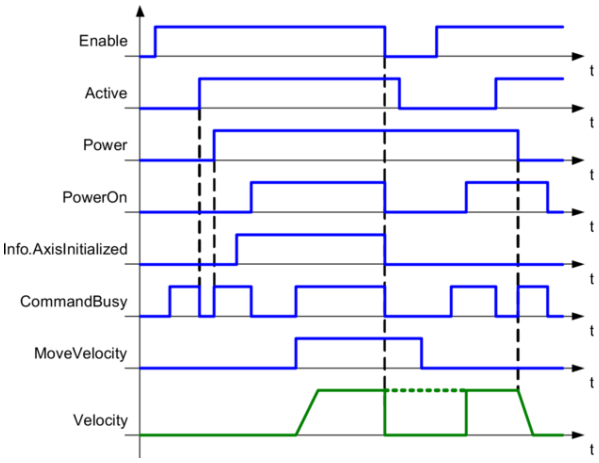


Figure 14: Timing diagram: Effect of the "Enable" input on other commands and status outputs

Application layer - mapp technology \ Components \ Mechatronics \ MpAxis - individual and multi-axis controllers \ Function blocks \ Timing diagrams

Status information

In the event of an error, the value of the "Error" output becomes = TRUE. The "StatusID" output contains numerical information that can be searched for in the Automation Studio help system. Additional information about the current status is made available in the "Info" output structure. This can be displayed when using WebXs in the browser.

The image below shows the MpAxisBasic component with an expanded info structure in WebXs. The "Error" output = TRUE and a value is given on the "StatusID" output. The info structure indicates that a PLCopen error has occurred when performing the "MOVE_VELOCITY" command. The error code is 29206. If you look for this error number in the Automation Studio help system, you will find a description. In this case the error description points out that an attempt was made to start a movement even though the controller wasn't turned on yet.

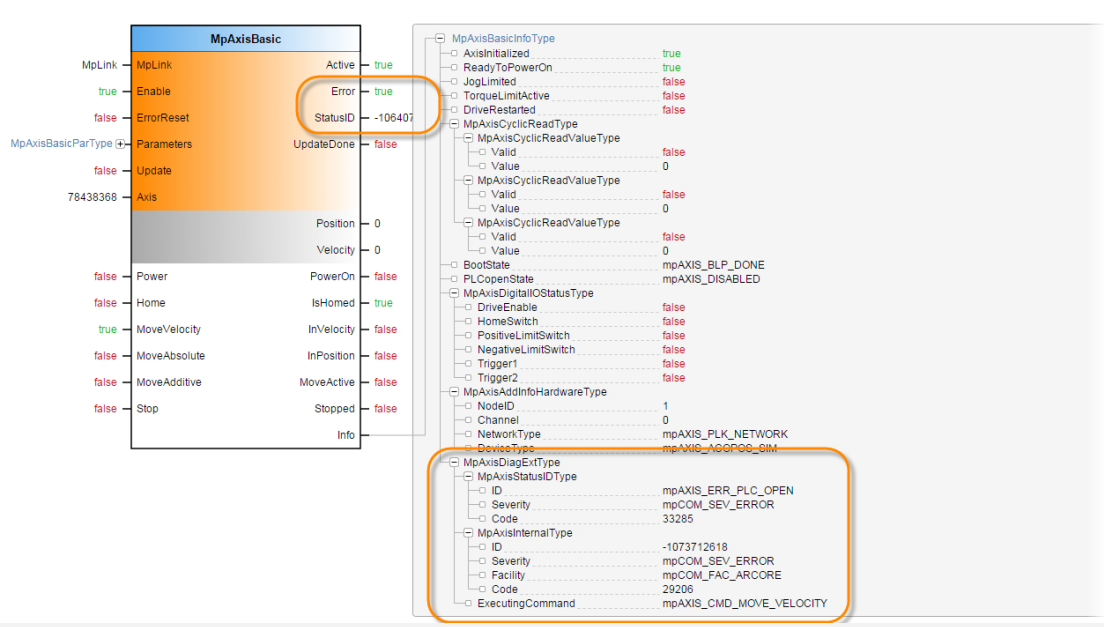


Figure 15: Comprehensive info structure - Representation in the WebXs: PLCopen error 29206 - "The controller is off"

In addition, an entry in the Automation Studio Logger named "$mapp" is generated. Additional details can be read in addition to the error number for identifying the cause of the error.

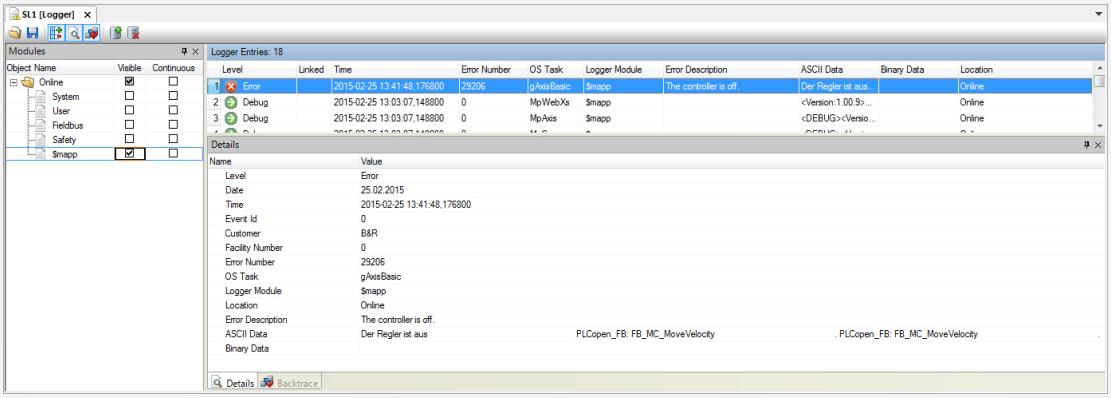


Figure 16: "$mapp" Logger file: PLCopen error 29206 - "The controller is off"

The cause of the error has been corrected and the error was acknowledged on the "ErrorReset" input. The controller is switched on again when a rising edge occurs on the "Power" input.

Exercise: Perform basic movements using the Watch window

Basic movements should be performed using the MpAxisBasic component. In the first step, the necessary component was already configured and called. Now declare command variables of data type BOOL and connect them with the MpAxisBasic inputs. The individual commands are executed first via the Watch window. The following commands should be used:

- "cmdPower" to switch on the controller
- "cmdHome" for homing the axis[3]
- "cmdErrorReset" to reset errors
- "cmdMoveVelocity" to move the axis using the speed setpoint
- "cmdMoveAdditive" to perform additive movements
- "cmdStop" to stop active movements

1) Declare command variables
2) Connect command variables with the MpAxisBasic inputs
3) Preconfigure the speed (1000 units/second) and distance (1000 units) using basic parameters.
4) Switch on the controller ("cmdPower"), wait for the "PowerOn" output.
5) Home the axis ("cmdHome"), wait for the "isHomed" output.
6) Perform movements ("cmdMoveVelocity"), observe "InVelocity" output
7) Check the status outputs and status structure of MpAxisBasic.

3.5 Overview of drive states

Defined states are determined in the PLCopen standard for operating a drive. They serve as an overview of the state in the positioning application as well as in the processing of error situations.

Status	Description
Disabled	The drive controller is switched off.
Standstill	The drive is not currently executing a movement and is ready for positioning commands.
Homing	The drive is executing a homing procedure.
Errorstop	The drive is at a standstill after an error. (= error stop)
Stopping	The drive is stopping an active movement.
Discrete Motion	The drive is executing a movement to a target position. The movement therefore has a defined end.
Continuous Motion	The drive is executing a movement without a target position. The movement therefore has no defined end.
Synchronized Motion	The drive is coupled to another drive.

Table 2: Overview of the states in PLCopen Motion Control state diagram

[3] "mpAXIS_HOME_MODE_DIRECT" is used for the exercise.

The most important states of a positioning application are included in the PLCopen state diagram. These can be used for coordinating positioning sequences. The current PLCopen state of an axis can be read using the info structure of the MpAxisBasic component or using the MC_ReadStatus function block from the ACP10_MC library.

A virtual axis does not have a drive controller. Unlike a real axis, a virtual axis does not have a Disabled state. The NC object "virtual axis" starts in the Standstill state and therefore doesn't have to be switched on separately.

Programming \ Libraries \ Motion libraries \ ACP10_MC \ Concept \ State diagram

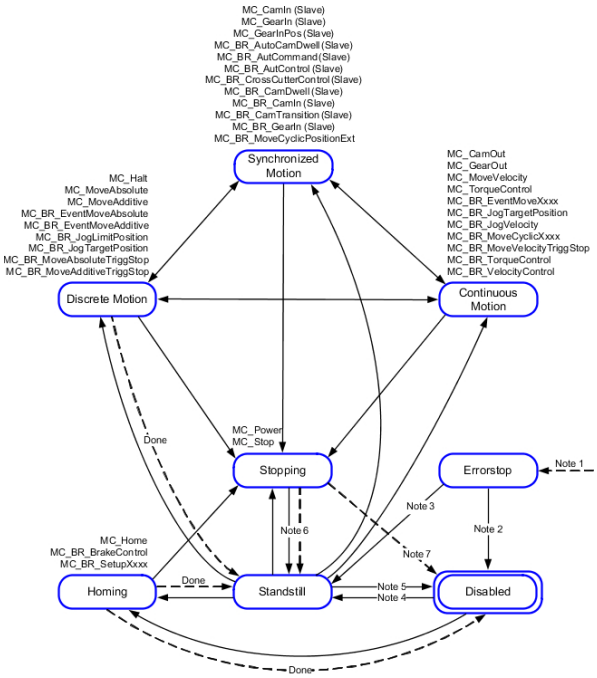


Figure 17: PLCopen Motion Control state diagram

The transitions between these states can be initiated by calling individual drive functions.

Let's assume that the axis is in the Standstill state. As soon as it has successfully performed a homing procedure, the "MoveAbsolute" command can be used to initiate a movement.

After the target position has been reached, the drive returns to its initial state. If a drive error occurs during positioning, then the axis enters the error state ("Error" input). This can be acknowledged using the "ErrorReset" input once the drive error has been corrected.

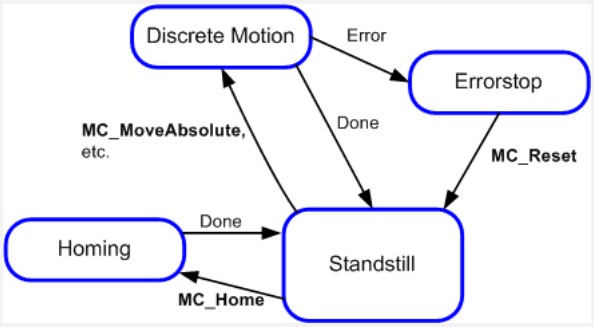


Figure 18: Sequence of states for performing an absolute movement

Exercise: Reading out the PLCopen state

The current PLCopen state can be determined using the info structure of MpAxisBasic. The info structure can be displayed in the Watch window or in WebXs.

Switch on the controller and perform a homing procedure. After every action, check the current PLCopen state.

1) Insert the MpWebXs library
2) Transfer the project
3) Check the info structure in the Watch window as well as in WebXs

Exercise: Forcing a drive error, deleting a drive error

Attempt to start a basic movement with a turned off controller or to exceed the maximum lag error. Subsequently observe the StatusID and examine the value in the Automation Studio help system. Subsequently acknowledge the error via the "cmdErrorReset" input and observe the status outputs.

1) Trigger a drive error
2) Check the "Error" output and the "StatusID"
3) Check the info structure
4) Acknowledge the error via the "ErrorReset" input
5) Check the "Error", "StatusID" and the info structure

Exercise: Automatically switch on the controller and perform a direct homing procedure

The existing program should now be somewhat automated. The MpAxisBasic component should be enabled immediately after switching on the controller. In addition, the controller should be automatically switched on after successfully enabling the component. Homing will then be carried out immediately. Additional commands are only allowed to be issued to the basic axis component in this state. In the event of an error, all commands should be reset so that the controller can be switched on again once the acknowledgment is complete and so that all commands for starting movements are disabled.

1) Set up state machine and project the necessary steps[4]
2) Set "cmdPower" if the "Info.ReadyToPowerOn" output is = TRUE
3) Set "cmdHome" if the "PowerOn" output is = TRUE
4) Reset "cmdHome" if "IsHomed" output is = TRUE
5) Reset all commands if the "Error" output is = TRUE

The drive is now prepared for performing basic movements. The program can optionally be extended so that the closed loop controller is automatically switched on and homed again after "cmdErrorReset" has been successfully executed.

[4] It is recommended to use high-level programming languages when setting up a state machine. When using graphic programming languages such as Ladder Diagram, MpAxisBasic status information can be used for approving commands.

4 Configuration management

There are different configuration access types when implementing mapp technology. The drive parameters can be saved in a NC Init module and be assigned to the drive via the NC mapping table. As an alternative to the NC Init module, mapp technology offers comfortable configuration management for the drive axis. It is managed in the Configuration View and is written to the drive when initializing the mapp component.

4.1 Axis initialization and configuration

Initialization using the mapp configuration

When added, each mapp component has a standard configuration. The NC Init module is reverted to in the default configuration of MpAxisBasic. The configuration is adjusted in the Configuration View. By changing the "Axis configuration" parameters to the value "Enabled", all drive parameters can be set in the configuration for the mapp component. This configuration is loaded to the drive after starting up the controller, enabling the MpAxis component and performing the first command (e.g. rising edge on the "Power" input).

In addition, a mapp configuration file is set up in the controller's memory. Changes can be made to the configuration during runtime with MpAxisBasicConfig (see 4.2 "Saving and loading the drive configuration" on page 23).

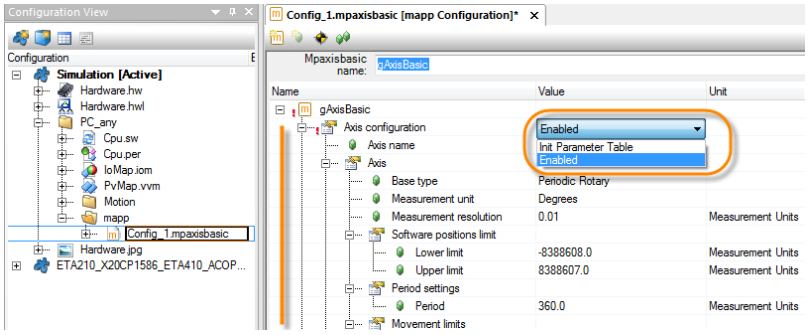


Figure 19: Enabling the configuration of the mapp component in the Configuration View

The example of a rotary table application will be used to explain the settings.

Task definition:

A pivoting carrier must move a product to different stations for processing (specific angular positions within a full rotation of 360°).

Positioning accuracy must be within 0.1°. The MpAxisBasic with the "MoveAbsolute" command function block will be used to approach the positions.

To make this procedure a little easier, the position is specified in degrees (with one decimal place):

```
BasicParameters.Distance := 135.0; (* Goal: 135° *)
```

The carrier is driven by a gear (gear ratio= 5:1) using a servo motor.

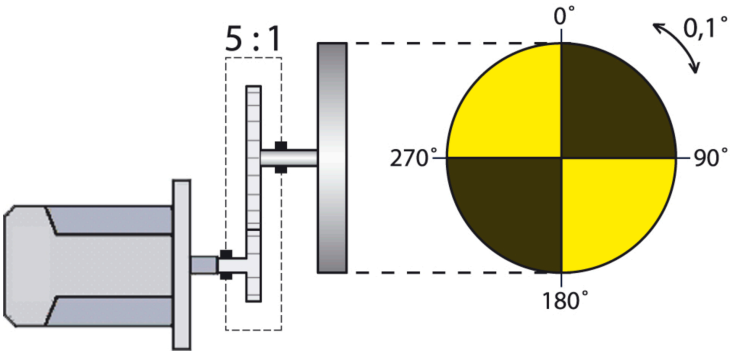


Figure 20: Sketch of the mechanical structure

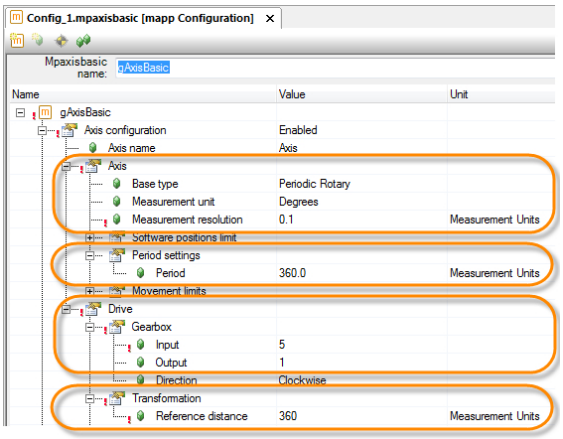


Figure 21: mapp configuration for the rotary table example

In the configuration of the axis, the unit of measurement, measurement resolution, gear ratio, axis period as well as a reference distance should be specified for a rotation on the gear output. The following configuration serves as a solution for the example:

The following relationships result when starting up with this configuration:

Degrees are used as the unit of measurement. The accuracy of the resolution is 0.1°. An axis period equals 360°. The gear ratio was specified as 5:1. A rotation on the gear output equals a rotation on the rotary table, which is the reason why the reference distance is also 360°.

Application layer - mapp technology \ Components \ Mechatronics \ MpAxis - individual and multi-axis controllers \ Technical information \

- ACOPOS axis initialization and configuration
- Units of measurement and axis scaling

Initializing via the NC Init module and NC mapping table

The NC Init module contains basic axis parameters used by NC Manager to initialize axis referencing when the controller is started.

This configuration is carried over from the MpAxisBasic component and automatically saved in the mapp configuration file. Changes are enabled at runtime via the MpAxisBasicConfig component.

Additional information: 7.3 "Setting position behavior and scaling" on page 34

Application layer - mapp technology

- Components \ Mechatronics \ MpAxis - individual and multi-axis controllers \ Technical information \ ACOPOS axis initialization and configuration
- Concept \ Component configuration

Motion \ Project development \ Motion control \ Configuration modules \ NC Init module

- NC Init module
- NC mapping tables

4.2 Saving and loading the drive configuration

Using the MpAxisBasicConfig component, the drive configuration, which is stored on the Flash memory in the form of a configuration file, is saved and loaded in the application. A data structure is connected to the component that is identically structured to the structure of the MpAxis drive configuration in the Configuration View. The same MpLink is used for MpAxisBasicConfig as for MpAxisBasic.

The configuration file is transferred to the drive after the first command (e.g. "Power" input) when enabling the MpAxisBasic component.

If the NC Init module is saved in the NC Test window and transferred to the controller, then the configuration parameters are automatically transferred to the mapp configuration file on the controller. If the configuration with MpAxisBasicConfig.Load is loaded, then the changed parameters are available for additional processing in the drive application.

Application layer - mapp technology \ Components \ Mechatronics \ MpAxis - individual and multi-axis controllers \ Technical information \ ACOPOS axis initialization and configuration

Exercise: Read the drive configuration

The objective of this exercise is to load the entire drive configuration. The procedure should be started using the "cmdLoadConfiguration" command. The proportional gain of the position controller should be determined for verification. The drive configuration is accessed using the MpAxisBasicConfig function block. The connection to the axis is established using the MpLink basic axis component.

1) Add MpAxisBasicConfig.
2) Declare the configuration structure.
3) Load the drive configuration using "cmdLoadConfiguration".

Exercise: Configuring the encoder interface and axis factor for a spindle drive

A gripper is positioned using a spindle drive connected to a motor via a gearbox. The gear ratio is 5:1 (i=5). For each motor rotation on the gearbox output, there is a feed rate of 0.6 mm. The positioning should be exact to 0.1 mm. The position setpoint is entered in mm*.

Determine the settings for unit of measurement, measurement precision, gear ratio and reference distance in the mapp configuration.

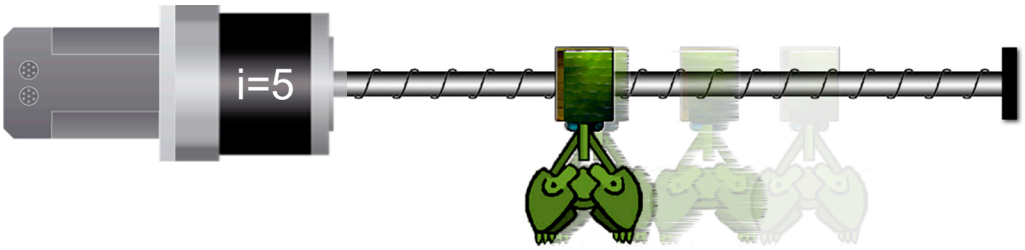
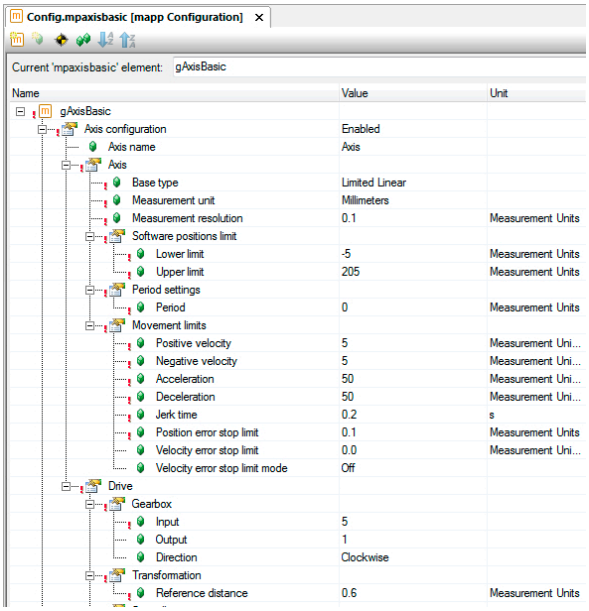


Figure 22: Schematic illustration of a spindle drive

1) Determine a suitable configuration for the units of measurement.
2) Determine the measurement precision.
3) Enter the gear ratio.
4) Enter the reference distance.
5) Define the movement limit values for this configuration and take into consideration the maximum speed of the motor, for example.
6) Configure the software limits in the range -5 mm to 205 mm.
7) Insert the MpAxisBasic component
8) Enable the MpAxisBasic component so that the configuration can be transferred to the drive
9) Perform the movements and check the results.

It should be taken into consideration that the parameters for maximum speed, lag error, acceleration and software limits should also be changed in the appropriate ratio when changing the axis parameter units.

* Entries such as 0.1 mm or 240 mm movement distance are permitted.



The following configuration results from the task definition:

1 mm is used as the measurement unit. The measurement resolution is 0.1 mm.

The software limits have been configured with -5 mm to 205 mm.

Movement parameters, such as acceleration and speed, also refer to the main unit of 1 mm.

A 0.6 mm feed rate occurs with a revolution on a gear output (reference distance).

If a speed of 1.2 mm/s is specified for a move command, for example, then 10 motor revolutions must be carried out per second drawing on the gear ratio.

Figure 23: mapp configuration for the spindle drive

If the configuration is carried out via the MpAxisBasicConfig function block, then the parameters should be assigned to the configuration structure as follows:

```
 // axis configuration (logical, PLCopen  units)
AxisConfiguration.Axis.BaseType := mpAXIS_LIMITED_LINEAR;
AxisConfiguration.Axis.MeasurementUnit := mpAXIS_UNIT_mm; (*unit 1 mm*)
AxisConfiguration.Axis.MeasurementResolution := 0.1; (*min. Value = 0.1 mm*)
AxisConfiguration.Axis.PeriodSettings.Period := 0;
 // software position limits
AxisConfiguration.Axis.SoftwareLimitPositions.LowerLimit := -5; (*unit 1 mm*)
AxisConfiguration.Axis.SoftwareLimitPositions.UpperLimit := 205;(*unit 1 mm*)
// drive configuration (mechanical)
AxisConfiguration.Drive.Gearbox.Input := 5;
AxisConfiguration.Drive.Gearbox.Output := 1;
(*Reference distance: 0.6 mm per load revolution*)
AxisConfiguration.Drive.Transformation.ReferenceDistance := 0.6;
// movement limits
AxisConfiguration.Axis.MovementLimits.VelocityPositive := 5; (*5 mm/sec*)
AxisConfiguration.Axis.MovementLimits.VelocityNegative := 5; (*5 mm/sec*)
AxisConfiguration.Axis.MovementLimits.Acceleration := 50; (*50 mm/sec² *)
AxisConfiguration.Axis.MovementLimits.Deceleration := 50; (*50 mm/sec² *)
```

The following relationships result from the settings:

Entry - PLCopen unit movement distance	Number of motor revolutions
0.6 mm	5 revolutions
2.4 cm	20 revolutions
90 cm	750 revolutions

Table 3: Relationship between the input value of the movement distance and the number of motor revolutions

5 Programming tips

The application program is a way to represent an automatic sequence for controlling the drive. In this process, it is important to call the function blocks in the program in a clear manner. Furthermore, error events must be taken into consideration and responded to.

For a structured sequence in the application program, a state machine can be used for high-level languages. In visual programming languages, steps can be set, for example, to control program flow.

Figure 24: A detailed look at the application program

5.1 Uses of control structures

The function block offers inputs for controlling drive functions. Processes can be started at a defined point in the program using command variables. Status outputs and output parameters provide information about the current state of the respective function. The following information can be provided about this:

- Was the command executed successfully?
- If not, what error occurred?
- Status of the process:
  - Is the axis moving?
  - Did the axis reach the target position?
  - Was homing performed successfully?

This information can be used for controlling the program sequence in the drive application. The program will have to respond differently depending on whether or not an error occurs.

The control structure that is especially suited to managing this kind of function processes is the step switching mechanism (state machine).[5]

This type of structure allows the implementation of individual steps whose sequence can be determined using a step index.

The diagram shows one possible design for this type of control structure.

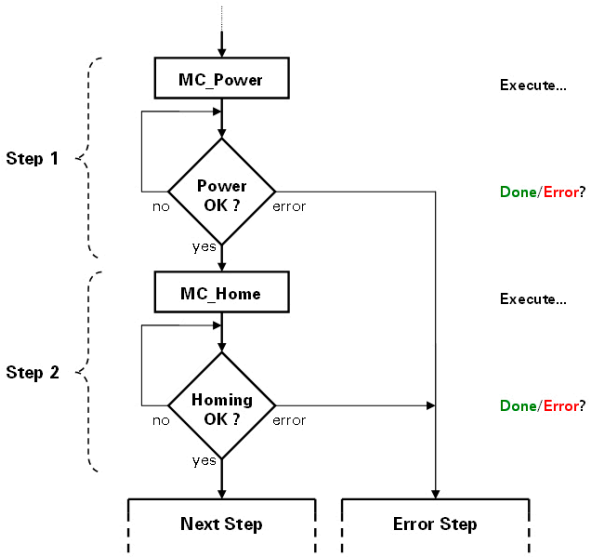


Figure 25: Sample control structure, structured programming

[5] All high-level programming languages have control structures that enable the implementation of a state machine or a step switching mechanism. In graphic programming languages, functions are enabled via logical links of several digital signals.

The necessary commands (switching on the controller, homing the axis, etc.) can be performed in the individual steps of the control structure. Status parameters such as Error, StatusID and the info structure can be used to determine the step where the application will continue.

This gives the application a clear structure and opens it up for future expansion.

5.2 Error indicators in the application program

There are basically several interfaces for troubleshooting mapp components (see 2.2 "Diagnostic options for mapp technology components" on page 7).

Error handling can be programmed into the drive application using the Error, StatusID and Info outputs. Error number descriptions can be found in the Automation Studio help system.

Figure 26: Evaluating errors in the application program

Application layer - mapp technology \ Concept \ Component design \ Inputs and outputs

Logger

Detailed data about the cause of the error is entered in the "$mapp" Logger file automatically. The Logger file can be opened using Automation Studio and the System Diagnostics Manager and be saved on the PC. Alternatively, the Logger file can be read using the "AsEventLog" library.

The mapp component MpComLoggerUI makes it possible to display events in the visualization application. It provides filters for limiting logger entires to certain mapp components, error numbers and message types.

By using the HMTL control for Visual Components, it is possible to embed the SDM page, which displays the Logger, directly into the machine visualization application.

Application layer - mapp technology \ Diagnostics

Application layer - mapp technology \ Components \ Infrastructure \ MpCom \ Function blocks \ MpComLoggerUI

Diagnostics and service \ Diagnostic tool

- Logger
- System Diagnostics Manager

Visualization \ Visual Components VC4 \ Control reference \ HTML view

Programming \ Libraries \ Configuration, system information, runtime control

- AsArLog
- ArEventLog

MpAlarm component and Visual Components alarm system

Using the MpAlarm component, the predefined alarms of the MpAxis component can be forwarded to the Visual Components alarm system using the MpAlarmUI function block. The entries displayed there can be filtered by group, time and priority. Language switching can be used for the displayed texts (see training module TM640 – Alarm System, Trends and Diagnostics).

Application layer - mapp technology \ Components \ Infrastructure \ MpAlarm - Support for alarm management

Visualization \ Visual Components VC4 \ Shared Resources \ Alarm System

Exercise: Resetting command variables in the event of error

In the event of an error, all command variables of the application program must be reset automatically. When an error occurs, this is indicated via the "Error" and "StatusID" outputs.

1) Evaluation of the positive edge of the "Error" output
2) Resets "cmdPower", "cmdHome" and "cmdMoveAdditive" command automatically in the event of an error

Exercise: Reading the axis error with the Logger, acknowledging the axis error

For example, if an attempt is made to start a movement without turning on the controller, then an error state is triggered. The error state is indicated via the MpAxisBasic function block outputs. A detailed description of the cause of the error is logged in the "$mapp" Logger file. After diagnostics have been completed, the error should be acknowledged via the "ErrorReset" input.

1) Start the movement with a switched-off controller
2) Open Logger
3) Look for the error number description in the Automation Studio help system[6]
4) Analyze the cause of error and acknowledge the drive error

The "ErrorReset" function block input resets all errors and puts the axis into the "Disabled" state. After the acknowledgment is completed, it is possible to switch on the controller on a rising edge on the "Power" input again.

Exercise: Switching on the controller automatically after a successful ErrorReset

After all errors have been successfully reset, the controller should be automatically switched on again via the drive application.

[6] The help page with the error description is directly opened by marking the error entry and pressing the <F1> key in the Logger.

1) Evaluate the falling edge of the "Error" output
2) Switch on the controller again with the "cmdPower" command
3) Perform a homing procedure, if required

Exercise: Read axis errors with the MpComLoggerUI component

Use the MpComLoggerUI component to evaluate the mapp events entered in the logger.

1) Add the MpComLoggerUI component.
2) Connect the MpLink from the MpAxisBasic component.
3) Initialize Scope parameter with the value mpCOM_CONFIG_SCOPE_COMPONENT.
4) Declare and connect the UIConnect structure.
5) Transfer the changes.
6) Trigger errors and analyze the entries in the UIConnect structure.

   A PLCopen error occurs, for example, if you try to start a movement when the controller is not switched on.

Application layer - mapp technology \ Components \ Infrastructure \ MpCom \ Function blocks \ MpComLoggerUI

6 Additional MpAxisBasic functions

The MpAxisBasic components offer more functions in addition to those used for drive preparation and for performing basic movements. Besides the functions that are already used, torque limiting, jog mode, autotuning as well as the cyclical reading of drive data are also enabled.

Application layer - mapp technology \ Components \ Mechatronics \ MpAxis - individual and multi-axis controllers \ Description

Exercise: Reading drive data cyclically

Cyclic reading of drive data can be configured using the configuration structure of the MpAxis component. Configure the component in such a way that the lag error is read cyclically from the drive.

1) Enable the entry for the cyclic reading of the lag error
2) Transfer the changes to the controller
3) Switch on a drive and perform movements
4) Check lag errors and record them with the Automation Studio trace if need be

Exercise: Implementing torque limiting

Specifying torque limiting in drive control takes place directly via the parameter structure on the MpAxis-Basic input. Specify, for example, a torque limit of 0.5 Nm. When the maximum torque is exceeded by 20% then an error should be output.

1) Transfer the torque limit
2) Transfer the threshold value for monitoring maximum values
3) Switch on the drive, perform a homing procedure
4) Start a movement
5) Enable torque by enabling the "TorqueLimit" input
6) Configure the behavior of the drive controller in the case of torque limiting and in the event of an error

The optional parameters of function blocks can be displayed in the Automation Studio help system by enabling the checkbox above the image of the function block instance.

Exercise: Performing autotuning and saving the configuration

The MpAxis component offers the function for automatically identifying parameters for the drive controller. Use the autotuning function to identify controller parameters for speed controllers and position controllers. Take into consideration that the axis is not permitted to move during initial parameter identification. It is necessary to perform movements for additional identification procedures, such as determining parameters for the feed-forward control.

1) Mode for autotuning configuration
2) Select maximum current, maximum speed and distance for the tuning procedure
3) Transfer the changes to the controller
4) Start the tuning procedure using the "MpAxisBasic.Autotune" input
5) Check the results

Expand the drive application so that you can save the calculated parameters in the drive configuration.

7 The PLCopen motion library (ACP10_MC)

The PLCopen motion library is listed in Automation Studio as the **ACP10_MC** library. It contains standardized PLCopen function blocks to control B&R drive solutions.

Basic functions such as homing or basic movements are defined in the ACP10_MC library. This provides the user with a uniform software interface for a specific area of standard applications.

The actual range of B&R drive solution functions goes far beyond that which is contained in the standard.

For example, expanded basic positioning and powerful function blocks for coupling axes are available as a complement to the standard functions[7].

Figure 27: PLCopen Motion Control logo

The ACP10_MC library has been expanded to include additional B&R-specific functions that allow the user to take advantage of these functionalities in a uniform manner. These advanced functions are operated in compliance with the PLCopen standard.

7.1 Function groups

The ACP10_MC library has a large number of function blocks. They can be divided into several groups according to their use and function.

Basic functions:

- Drive preparation
- Basic movements such as additive and absolute movements
- Jog mode
- Determining the status of the drive
- Reading set and actual values
- Determining and acknowledging drive errors
- Position measurement
- Managing PLCopen axis parameters

Figure 28: Overview of basic functions

Making changes to movements, drive configurations and data:

- Torque control
- Trace functions
- Cyclic setpoint generation
- Managing axis parameters

[7] A detailed overview of multi-axis and coupling functions is provided in the training module "TM441 – Motion Control: Multi-axis Functions".

Multi-axis functions:

- Creating electronic gears
- Phase and offset shifting
- Connecting drives using cam profiles
- Configuring and controlling the Cam Profile Automat

Technology functions:

- Registration mark control
- Cutting units
- Cam Profile Automat

Programming \ Libraries \ Motion libraries \ ACP10_MC \ Function blocks

Programming \ Libraries \ Motion libraries \ ACP10_MC \ Function blocks \ Overview of the supported function blocks

7.2 Compatibility of ACP10_MC and MpAxis

The MpAxis mapp component function blocks are based on the function blocks in the ACP10_MC library. The two libraries are completely compatible with each other. In both libraries, the axis is accessed via the axis reference.

Parameters and commands for the drive are transferred when commands are activated. The state in the PLCopen state diagram then changes. This structure makes it possible to add features to applications that have been programmed with MpAxisBasic using function blocks from the ACP10_MC library.

The function compatibility, for example, enables the controller to be switched on and homed with MpAxisBasic and to start a relative movement on the same axis with the MC_MoveAdditiv function block from the ACP10_MC library. The change of state from "Standstill" to "Discrete Motion" is shown immediately in the info structure of the MpAxisBasic component.

Exercise: Combine mapp technology and the APC10_MC library

The mapp components for controlling drives are based on the PLCopen standard. For this reason, all mapp components are compatible with the function blocks in the ACP10_MC library. The MpAxisBasic component offers all relevant functions; it is possible, however, to expand the motion application using function blocks from the ACP10_MC library. The drive data is accessed in both cases via the axis reference.

In this task, the drive should be prepared using the MpAxisBasic component and a movement should be performed using the MC_MoveVelocity function block from the ACP10_MC library.

1) Add the MC_MoveVelocity function block to the application.

2) Switch on the controller and perform a homing procedure with MpAxisBasic

3) Start a movement in the positive direction using MC_MoveVelocity.

4) Verify the state change using the MpAxisBasic outputs.

Programming \ Libraries \ Motion libraries \ ACP10_MC \ Function blocks \ Basis movements

7.3 Setting position behavior and scaling

It is possible to make adjustments to the position value using additional settings in the NC mapping table if the NC Init module is used for initializing the axis instead of the mapp configuration. With these settings, just as with the help of the mapp configuration, a periodical position behavior and a scaling of the position can be achieved.

Periodic position behavior is required applications such as turntables.

Configuration of the encoder interface

Motor rotations are scaled to units on the load in the encoder interface settings for the axis. The units on the load are referred to as axis parameter units. The values used are positive integers.

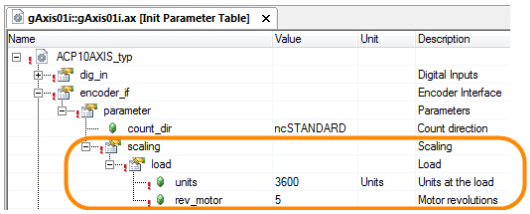


Figure 29: NC Init module: Setting the ratio of units to motor rotations

For example, in the image, 1000 axis parameter units are allocated to a motor revolution. That amounts to 1000 axis parameter units per revolution. This makes it possible to divide the revolutions according to the requirements of the application (keeping the maximum resolution of the encoder in mind). DINT is the data type for the position.

Setting of period and factor in the NC mapping table

All PLCopen-compliant function blocks use the REAL data type for position specifications.

In the NC mapping table via the entry PLCopen_ModPos="<Period>,<Factor>", the values for position period and axis factor are predefined for adjusting the position value.

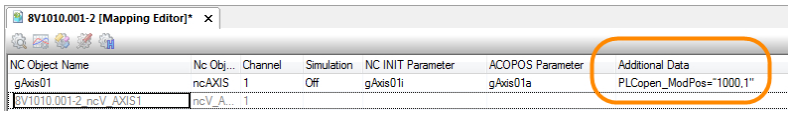


Figure 30: Advanced settings in the NC mapping table

Velocity and acceleration values in the PLCopen application also refer to this scaling.

Besides the use of the NC Init module, mapp technology offers expanded configuration management. This means that these configuration settings that were inserted in the Configuration View in the application software via a function block can be easily managed.

Additional information: 4 "Configuration management" on page 21

7.3.1 Axis factor

All PLCopen-compliant function blocks use the REAL data type for the axis position. This data type enables the use of decimal places, which in turn can make simplified scaling pretty interesting.

Scaling can also be referred to as conversion. The following example should help to illustrate this:

A certain application requires positioning to be accurate down to 1 µm. It's assumed that a movement distance of 1 µm per axis parameter unit has been configured through the corresponding setting of the encoder parameters.

It is advantageous for the positioning application if the position specifications can be given in millimeters for function blocks. This requirement can be fulfilled by configuring the axis factor.

The equation looks like this: $PLCopenUnits = \frac{AxisparameterUnits}{Factor}$

So if we set the factor to the value 1000, we will get our scaling to millimeters. If a movement with a distance of the value 1 is started via a function block, then the axis moves by 1000 axis parameter units. This corresponds to 1 mm or 1000 µm.

Axis factor	PLCopen units [REAL]	Axis parameter units [DINT]	Movement distance
1	1.0	1	1 µm
1000	1.0	1000	1 mm
1000	0.001	1	1 µm

Table 4: Comparing PLCopen units to positioning units depending on the set axis factor

Programming \ Libraries \ Motion libraries \ ACP10_MC \ Concept \ Implementation \ Axis factor

7.3.2 Axis period

For continuous axis movements, it is often necessary to determine a position value periodically. If the value for the period is set, then the position value is adjusted according to this entry. This value refers directly to scaling the axis in the encoder parameters. All function blocks then work with this periodic position.

For example, if the value for the axis period is set to 1000, then the position value always increases from 0 to 999 during a positive movement direction before being reset to 0 and increasing again to 999.

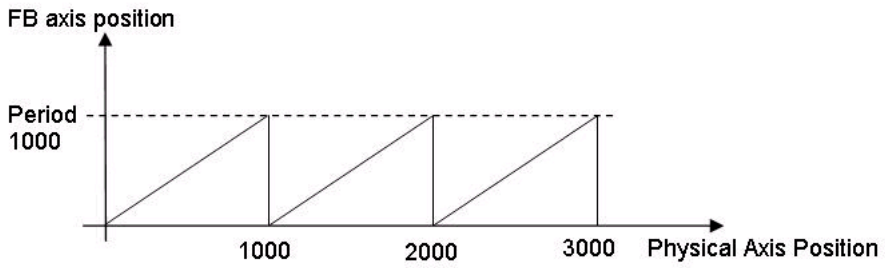


Figure 31: Periodic axis position: Axis period = 1000 PLCopen units

In this way, this behavior can be disabled by specifying the value 0 for the axis period if periodic behavior is not required but a specific factor still has to be used. In this case, the axis movement range is limited to ±8,388,608 units ($\pm 2^{23}$)[8].

Programming \ Libraries \ Motion libraries \ ACP10_MC \ Concept \ Implementation \ Axis period

The axis parameter units are always recorded in NC Trace when movements are recorded. Acceleration and speed also refer to axis parameter units.

Exercise: Configuring a rotary axis

This simple example will implement the settings already described and demonstrate the resulting possibilities.

Task definition:

A pivoting carrier must move a product to different stations for processing (specific angular positions within a full rotation of 360°).

Positioning accuracy must be within 0.1°. The MpAxisBasic with the "MoveAbsolute" command function block will be used to approach the positions.

To make this procedure a little easier, the position is specified in degrees (with one decimal place):

```
BasicParameters.Distance := 135.0; (* Goal: 135° *)
```

The carrier is driven by a gear (gear ratio= 5:1) using a servo motor.

[8] The limitation is confined to this value because this is the largest number which can be displayed by the REAL data type without losing accuracy.

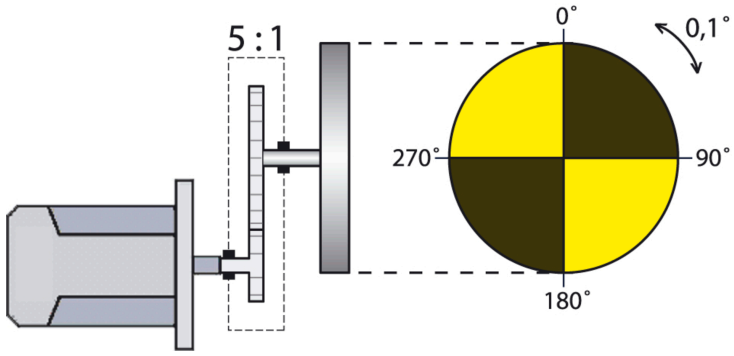


Figure 32: Sketch of the mechanical structure

1) Determine the suitable encoder settings for the NC Init module.
2) Expand the settings for axis period and axis factor in the NC mapping table.
3) Add the "cmdMoveAbsolute" command to the positioning application.
4) Test the carrier positioning for different angles within a revolution.

Possible solution:

First, the basic settings for the encoder interface are made in the NC Init module. For the position resolution in 0.1° steps, 3600 units are required for one revolution of the rotating carrier. These units are distributed over 5 motor revolutions because of gear ratio reduction. If a movement of 3600 units is executed, then the motor performs exactly 5 revolutions and the carrier is rotated exactly 360°. The gear is thus already included.

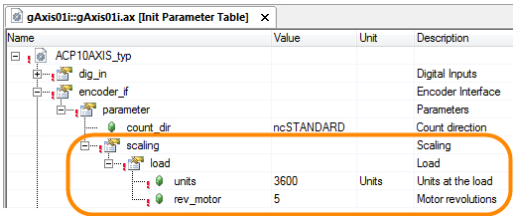


Figure 33: Encoder interface settings in the Init parameter module

In order to get the desired scaling for the positioning function, the axis period and the axis factor have to be configured. For this, an axis factor of 10 and an axis period of 3600 have to be set (refers directly to the settings on the encoder interface).

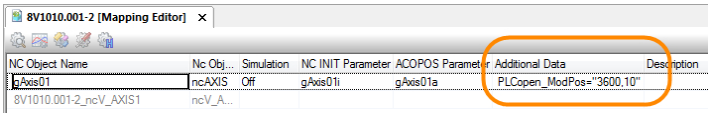


Figure 34: Advanced settings for the periodic axis in the NC mapping table

Axis factor	PLCopen units [REAL]	Axis parameter units [DINT]	Movement distance Carrier	Angle of rotation Motor
1	1.0	1	0.1°	0.5°

Table 5: Comparing PLCopen units to positioning units depending on the set axis factor

Axis factor	PLCopen units [REAL]	Axis parameter units [DINT]	Movement distance Carrier	Angle of rotation Motor
10	1.0	10	1°	5°
10	72,0	720	72°	360°
10	135,0	1350	135°	675°
10	360,0	3600	360°	1800°

Table 5: Comparing PLCopen units to positioning units depending on the set axis factor

With reference to the example shown, an additive movement of 72 PLCopen units means a position increase in NC Trace of 720 axis parameter units.

A traverse path of 720 PLCopen units means a traverse path of 2 axis periods in the example configuration (equivalent to two revolutions on the rotating carrier) as well as a position increase of 7200 axis parameter units in NC Test.

The current PLCopen position (based on the previous homing procedure to position 0) is subsequently 0. NC Trace shows the real position with the value 7200. This corresponds to 10 motor revolutions

8 Summary

The powerful MpAxis component is available for controlling drive functions. The function blocks contained comply with the PLCopen standard and are set apart by their efficient design and usability.

Integration in the controller application begins as soon as the user is confident with mapp technology (which is quite simple). Elaborate configuration management, web-based diagnostics, the direct integration of error information in the Automation Studio Logger window as well as in the System Diagnostics Manager are available through the component architecture. Incorporating drive alarms in the visualization application is enabled by the MpAlarm component.

Figure 35: mapp technology offers a comprehensive portfolio of functions

The enhancement of the drive application is possible because of function compatibility with the other Motion Control libraries, for example, with the function blocks of the ACP10_MC and the MT_LoadSim libraries.

9 Solutions

9.1 Solution: Automatically switch on the controller and perform a direct homing procedure

```
(*********************************************************************
 * COPYRIGHT -- Bernecker + Rainer
 *********************************************************************
 * Program: mp410_axis
 * Created: October 2014
 *********************************************************************
 * Implementation of program mp410_axis
 *********************************************************************)

PROGRAM _INIT

    (*set basic paraemters*)
    BasicParameters.Velocity := 1000; (*1000 units/second*)
    BasicParameters.Distance := 1000; (*1000 units distance*)

END_PROGRAM

PROGRAM _CYCLIC

    CASE sStep OF
        enINIT:

        (*do nothing; sequence is started manually*)

        enSTART:

            MpAxisBasic_0.MpLink :=         ADR(gMpLink_AxisBasis_gAxis1);
            MpAxisBasic_0.Axis :=             ADR(gAxis01);
            MpAxisBasic_0.Enable :=         1;
            MpAxisBasic_0.Parameters :=     ADR(BasicParameters);

            (*is axis ready to be powered on?*)
            IF MpAxisBasic_0.Info.ReadyToPowerOn = TRUE THEN
                sStep := enPOWER_ON;
            END_IF

        enPOWER_ON:

            cmdPower := TRUE;

            (*axis is powered on?*)
            IF MpAxisBasic_0.PowerOn = TRUE THEN
                sStep := enHOME;
            END_IF

        enHOME:
```

```
            cmdHome := TRUE;

            (*axis is homed?*)
            IF MpAxisBasic_0.IsHomed = TRUE THEN
                sStep := enOPERATION;
                cmdHome := FALSE;
            END_IF

        enOPERATION:

            (*commands basic movements*)
            cmdMoveVelocity;
            cmdMoveAdditive;
            cmdStop;

            (*reset command cmdMoveAddive when position is reached*)
            IF EDGEPOS(MpAxisBasic_0.InPosition) = TRUE THEN
                cmdMoveAdditive := FALSE;
            END_IF

            (*update parameters*)
            cmdUpdate;

            (*axis in error step?*)
            IF MpAxisBasic_0.Error = TRUE THEN
                sStep := enERROR;
            END_IF

        enERROR:
            (*power off*)
            cmdPower := FALSE;

            (*reset all commands*)
            cmdMoveVelocity := FALSE;
            cmdMoveAdditive := FALSE;
            cmdStop := FALSE;
            cmdUpdate := FALSE;

            (*implement error handling here*)
            cmdErrorReset;
            IF MpAxisBasic_0.Error = FALSE THEN;
                sStep := enSTART;
                cmdErrorReset := FALSE;
            END_IF

    END_CASE

    (*execute commands*)
    MpAxisBasic_0.Power :=            cmdPower;
    MpAxisBasic_0.Home :=            cmdHome;
    MpAxisBasic_0.ErrorReset :=     cmdErrorReset;
    MpAxisBasic_0.MoveVelocity :=     cmdMoveVelocity;
```

```
    MpAxisBasic_0.MoveAdditive :=     cmdMoveAdditive;
    MpAxisBasic_0.Stop :=             cmdStop;
    MpAxisBasic_0.Update :=         cmdUpdate;

    (*call all mapp components*)
    MpAxisBasic_0();

    (*movement with MC_MoveVelocity so watch status changes of component MpAxisBasic*
    MC_MoveVelocity_0.Axis := ADR(gAxis01);
    MC_MoveVelocity_0.Execute := cmdMoveVelocity_ACP10_MC;
    MC_MoveVelocity_0.Velocity := 1000;
    MC_MoveVelocity_0.Acceleration := 2000;
    MC_MoveVelocity_0.Deceleration := 2000;
    MC_MoveVelocity_0.Direction := mcPOSITIVE_DIR;

    (*call all ACP10_MC function blocks*)
    MC_MoveVelocity_0();

END_PROGRAM

PROGRAM _EXIT

    (*disable all mapp components*)
    MpAxisBasic_0.Enable := FALSE;
    MpAxisBasic_0();

END_PROGRAM
```

Seminars and training modules

Automation Studio seminars and training modules

Programming and configuration	Diagnostics and service
SEM210 – Basics SEM246 – IEC 61131-3 programming language ST* SEM250 – Memory management and data storage SEM410 – Integrated motion control* SEM441 – Motion control: Electronic gears and cams** SEM480 – Hydraulics** SEM1110 – Axis groups and path-controlled movements** SEM510 – Integrated safety technology* SEM540 – Safe motion control*** SEM610 – Integrated visualization*	SEM920 – Diagnostics and service for end users SEM920 – Diagnostics and service with Automation Studio SEM950 – POWERLINK configuration and diagnostics* If you don't happen to find a seminar on our website that suits your needs, keep in mind that we also offer customized seminars that we can set up in coordination with your sales representatives: SEM099 – Individual training day Please visit our website for more information****.****: **www.br-automation.com/academy**

Overview of training modules	
TM210 – Working with Automation Studio TM213 – Automation Runtime TM223 – Automation Studio Diagnostics TM230 – Structured Software Development TM240 – Ladder Diagram (LD) TM241 – Function Block Diagram (FBD) TM242 – Sequential Function Chart (SFC) TM246 – Structured Text (ST) TM250 – Memory Management and Data Storage TM400 – Introduction to Motion Control TM410 – Working with Integrated Motion Control TM440 – Motion Control: Basic Functions TM441 – Motion control: Electronic gears and cams TM1110 – Integrated Motion Control (Axis Groups) TM1111 – Integrated Motion Control (Path Controlled Movements) TM450 – Motion Control Concept and Configuration TM460 – Initial Commissioning of Motors TM500 – Introduction to Integrated Safety TM510 – Working with SafeDESIGNER TM540 – Integrated Safe Motion Control	TM600 – Introduction to Visualization TM610 – Working with Integrated Visualization TM630 – Visualization Programming Guide TM640 – Alarm System, Trends and Diagnostics TM670 – Advanced Visual Components TM920 – Diagnostics and service TM923 – Diagnostics and Service with Automation Studio TM950 – POWERLINK Configuration and Diagnostics TM280 – Condition Monitoring for Vibration Measurement TM480 – The Basics of Hydraulics TM481 – Valve-based Hydraulic Drives TM482 – Hydraulic Servo Pump Drives TM490 – Printing Machine Technology In addition to a printed version, our training modules are also available on our website for download as electronic documents (login required): Visit our website for more information: **www.br-automation.com/academy**

Process control seminars and training modules

Process control standard seminars	Process control training modules
SEM841 – Process Control Training: Basic 1 SEM842 – Process Control Training: Basic 2 SEM890 – Advanced Process Control Solutions	TM800 – APROL System Concept TM811 – APROL Runtime System TM812 – APROL Operator Management TM813 – APROL XML Queries and Audit Trail TM830 – APROL Project Engineering TM890 – The Basics of LINUX Visit our website for more information: **www.br-automation.com/academy**

* SEM210 - Basics is a prerequisite for this seminar.
** SEM410 - Integrated motion control is a prerequisite for this seminar.
*** SEM410 - Integrated motion control and SEM510 - Integrated safety technology are prerequisites for this seminar.
**** Our seminars are listed in the Academy\Seminars area of the website.
**** Seminar titles may vary by country. Not all seminars are available in every country.