Motion Control: Basic Functions

TM440

Requirements

Training modules:	TM410 – The Basics of ASiM
Software	Automation Studio 3.0.90 Automation Runtime 3.08 ACP10_MC Library 2.280
Hardware	None

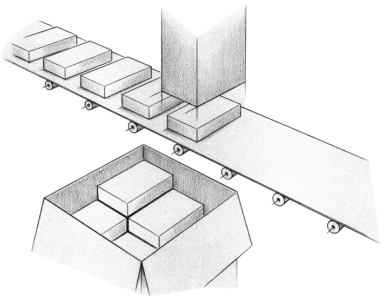
TABLE OF CONTENTS

1 INTRODUCTION

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

This is where commands are defined and transferred to the drive and where signals from the process are evaluated. The program controls the drives used in the process with an automatic sequence.

A solid overview of the available tools is the foundation for setting up a positioning task. Therefore, the first step is to obtain a clear overview of the entire range of function blocks used for controlling drives.



Palletizing

PLCopen Motion Control logo

In this training module we will also take a look at using and integrating positioning functions in an 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.

The PLCopen Motion Control standard plays a central role in the operating concept of B&R's drive solution.

1.1 Training module objectives

In this training module, you will be given an overview of how to apply and integrate the PLCopen library.

You will receive information about ...

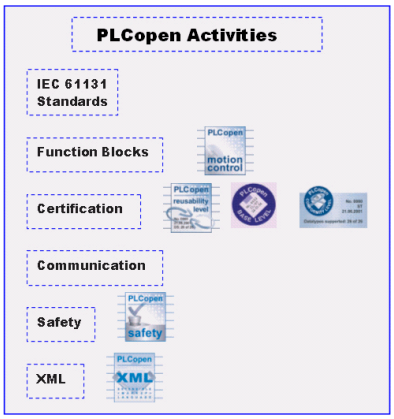
- ... the advantages of using PLCopen functions
- ... integrating PLCopen functions into your project
- ... the structure and functionality of function blocks
- ... the basic functions for preparing the drive and carrying out standard movements
- ... the typical structure of application programs
- ... the process for adapting the sample programs

2 THE PLCOPEN STANDARD

2.1 General information

The role of software is continually increasing in automation systems. With a wider range of functions, the complexity of applications also increases, making software development and maintenance that much harder. In addition, there are a number of solutions (→ products) on the market to choose from.

The PLCopen organization is busy standardizing different areas, components, and tools in the field of industrial automation. As a result, there are different areas of activity.



Areas of activity for PLCopen

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

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

All automation solution suppliers who belong to this organization provide a uniform software interface for their system that is defined using PLCopen. B&R is an active member.

2.2 Advantages

Software development independent of manufacturer

The application program is hardware-independent because of its uniform operation and standard positioning functions.

Reduced development times

Getting to know the specific system solution from a particular provider is no longer necessary. In principle, this allows you to start implementing basic functionalities right away. In addition, the PLCopen standard is distinguished by its high degree of usability.

Simple program maintenance

With PLCopen, you have a tool that uses simple function blocks to provide a clearly arranged structure in the application program. This makes it much easier to handle corrections, perform updates and find errors.

Fully supported by B&R

The PLCopen standard for positioning applications is available for B&R drive solutions. This means that project setup and configuration can be carried out quickly and easily thanks to the use of standardized function blocks.

In addition to standardization, B&R-specific functions are also available that have been prepared in accordance with the standard. This makes it possible to use the entire functional range of the B&R drive solution in a uniform manner.

These function blocks are provided in the ACP10_MC library.

Programming \ Libraries \ Motion libraries \ ACP10_MC \

3 THE PLCOPEN MOTION LIBRARY

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

In addition to these standardized function blocks, this library also includes B&R-specific expansions.

3.1 Structure and composition

The ACP10_MC library contains basic functions such as "Positioning to an absolute target position" or "Homing procedure" that can be used with any system.

This provides the user with a completely uniform software interface for a specific area of standard applications.

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

PLCopen Motion Control logo

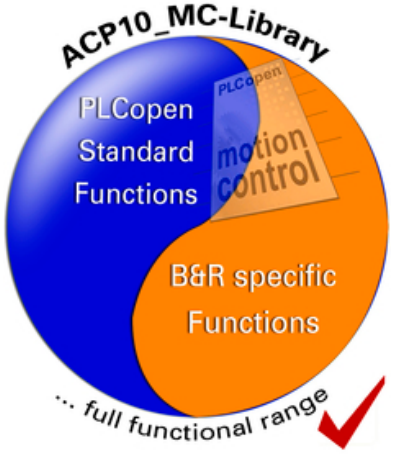
For example, powerful tools for connecting drives are provided (see TM441 Motion Control: ASiM Multi-axis Functions). Basic positioning functions are also available with advanced options.

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 exactly the same as the standard functions.

Whether a function block is a PLCopen standard function block or a B&R-specific function block is indicated by its name:

Standard function blocks always include the prefix "**MC_**" at the beginning of the name, e.g. MC_MoveAdditive or MC_ReadAxisError.

B&R-specific function block and expansion names begin with "**MC_BR_**", e.g. MC_BR_BrakeOperation or MC_BR_AutControl.



ACP10_MC library

3.2 Function groups

The ACP10_MC library contains a relatively large number of function blocks, which can be roughly divided into several groups according to their use and function.

Basic functions for:

- Preparing the drive
- Standard movements such as additive and absolute movements
- Determining drive status
- Reading set and actual values
- Determining and acknowledging drive errors
- Query and control functions for digital I/O signals
- Position measurement
- Managing PLCopen axis parameters
- Managing parameter IDs

Overview of basic functions

Multi-axis functions for:

- Creating an electronic gearbox
- Connecting drives using cam profiles
- Configuring and controlling the cam profile automat

Technology functions for:

- Torque control
- Cyclic set value inputs
- Print mark control
- Cam profile automats

Programming \ Libraries \ Motion libraries \ ACP10_MC \ Function blocks

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

4 INTEGRATION IN THE CONTROL PROJECT

The process for creating a drive configuration in Automation Studio is described in the preceding training module (TM410). Specific commands may already have been transferred to a drive using the "NC Test" diagnostics tool.

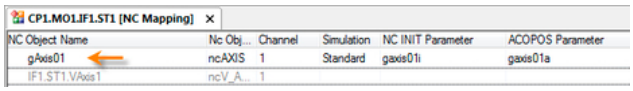
The following example shows how an application program for controlling axis movements can be structured.

An axis reference for accessing the axis object will be needed first. This axis reference has already been generated by the Motion Wizard and added to the mapping table.

Motion \ Configuration \ Motion control \ Creating an axis

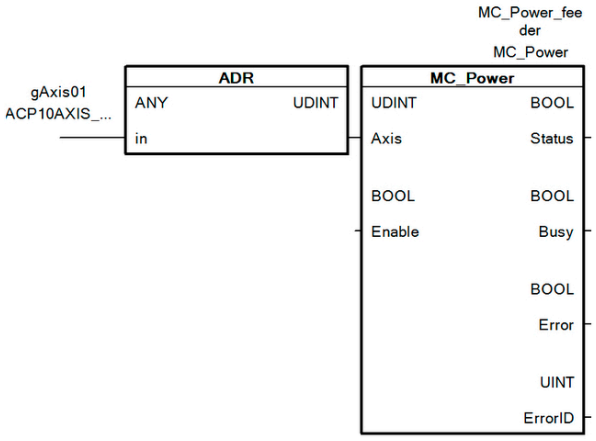
4.1 Using the axis reference

The wizard creates a global PV of type ACP10AXIS_typ with the same name that is listed in the NC mapping table. The connection is then made from the software object to the actual hardware via POWERLINK.



NC mapping, mapping table

The address of the axis reference is passed to all PLCopen function blocks using the "Axis" parameter. In Ladder Diagram, the address contact can be used as an alternative to the address function.



The axis reference is passed to the MC_Power function block

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

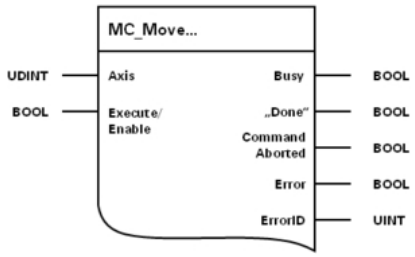
5 APPLICATION BASICS

This section provides some basic information about using ACP10_MC function blocks. We will look at how to operate the function blocks as well as possibilities available for monitoring procedures.

5.1 Controlling the function blocks

All function blocks are accessed using uniform operating and status parameters.

This standardized operation of all function blocks makes it easier to use the program and helps to ensure a clear overview during programming.



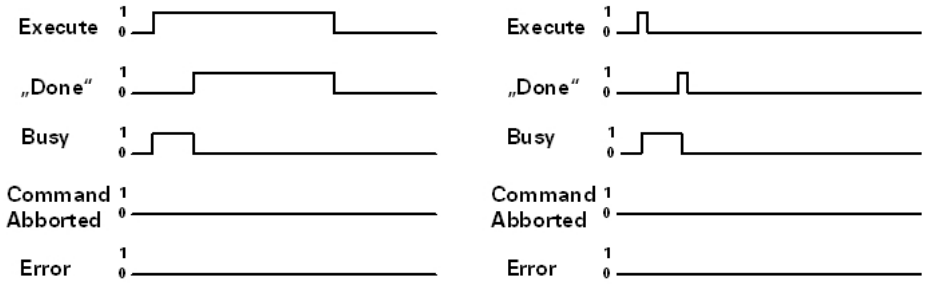
Standard function parameters

Parameters	Description
Axis	Specifies the axis reference to determine the axis for which the function block should be used.
Enable	Function blocks with an enable input control tasks according to the level on this input. The values on the function block inputs are taken over when it is set to TRUE. When the input is set to FALSE, an action is also carried out and all outputs are reset.
Execute	Function blocks with an "Execute" input read their input values and execute their tasks one time when a positive edge occurs on the "Execute" input. (For example, a new set speed is taken over when a positive edge occurs on Execute). If a function is started with this input, it cannot be ended prematurely by resetting the input. Another block must be called to interrupt or stop it. (For example MC_Halt, MC_Stop, MC_Move, etc.)
Busy	Indicates that the respective action has been successfully started and is currently being executed. As long as the output is set, the function block must continue to be called, otherwise the status information, function, etc. is negatively affected.
"Done"	Each function block contains a status input that indicates the successful completion of the action. This output is labeled differently depending on the function block, but always serves the same purpose.
CommandAborted	Signals that the command was aborted by another function block call.
Error	Signals that an error occurred during the function block call.
ErrorID	If an error occurs, the corresponding error number is returned here. This provides information about the cause of error A listing of the error numbers can be found in the Automation Studio online help.

Table: Description of standard parameters and status information

Programming \ Libraries \ Motion libraries \ ACP10_MC \ Function blocks \ Error numbers

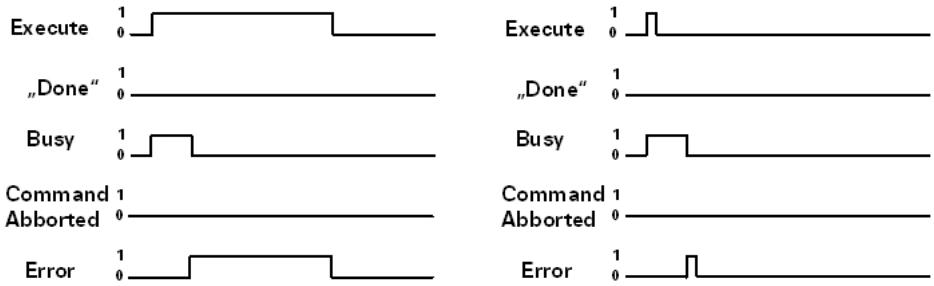
Successful command processing



Command processing successful

The process is started when a positive edge occurs on the Execute input. Active processing of a command is indicated via the Busy output. Done (or InVelocity, InGear, etc.) indicates that the action has been completed successfully.

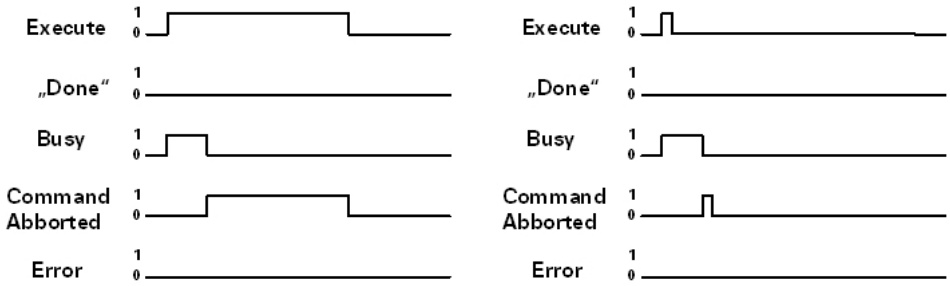
Command processing with error



Command processing failure

In this case, an error occurs after enabling the action. If the "Error" output is active, then a corresponding error number is indicated using "ErrorID".

Command processing interrupted



Command processing aborted

In this case, the command that is about to be processed is interrupted by another command. This is always the case when the executed function block affects an already active function block Busy= 1).

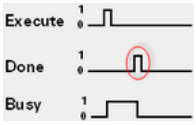
For example, an absolute movement (MC_MoveAbsolute) can be interrupted by a relative movement (MC_MoveAdditive).

Function blocks for reading status data (position, parameter, etc.) and commands for controlling movements do not affect each other.

Summary

The status information Done, Command Aborted, Error and ErrorID remain the same until the Execute input is reset.

If the Execute input is already inactive before these signals arrive, the status outputs and error indicators are set for the duration of one cycle:



Signal duration is equal to one cycle

In order to reset the function block outputs if an error occurs, the Execute / Enable input must be set tot FALSE, the cause of the error must be corrected (and the error may have to be acknowledged) and then the function must be restarted.

The function blocks with an Enable input are only active as long as this input remains set. Otherwise the action is ended and the status values on the function block outputs are reset.

The function block calls for the PLCopen functions must be called in the cyclic section of the program. In high level language programs, we recommend calling the function at the end of the program for all instances.

For graphic programming languages, special care must be taken if networks are to be jumped over within the flow of the program.

The right approach can also be taken the sample programs (see 7 "Using the sample programs").

Exercise: Apply the basic ACP10_MC functions

The function blocks from the ACP10_MC library can be used quickly and easily in an application program.

The drive should be able to carry out an absolute movement. Use the ladder diagram programming language for this.

1) Add the axis using the Motion wizard
2) Add the MC_Power function block
3) Add the MC_Home function block
4) Add the MC_MoveAbsolute function block
5) Then send the axis reference to all function blocks
6) Transfer the program
7) Restart the CPU

8) Switch on the drive
9) Reference the drive
10) Trigger an absolute movement

Exercise: Read the current position

Read the current position of the drive with the MC_ReadActualPosition function block.

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

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

Programming \ Libraries \ Motion libraries \ ACP10_MC \ Function blocks \ Determination of Axis State

The axis reference should be linked to all function blocks. So you do not have to call the address function all the time, the address can be stored as an internal variable and then passed on to the function blocks. This makes it easy to replace the axis reference in the application program.

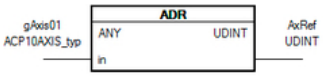
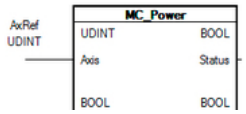
Init program	 Determine the address value for the axis reference
Cyclic program	 Transferring the axis reference

Table: Transferring the axis reference using an address variable

The MC_Power function block is required to switch on the controller and the power elements. On an ACOPOSmulti system, it is also required to switch on the power supply module.

The initialization value of the function block instances must be empty. Accordingly, no variables with the "permanent" or "retain" attribute may be used.

5.2 Overview of the drive states

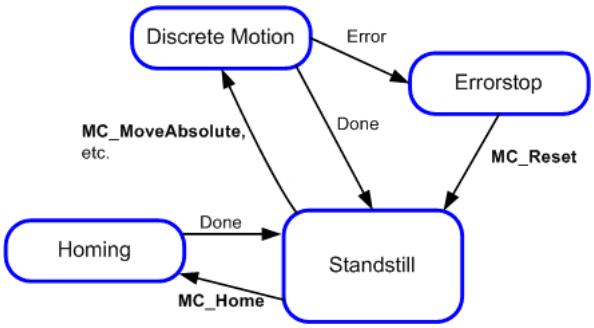
Defined states are determined for operating a drive. These states provide a simpler overview of complex movement procedures and make it easier to process error situations.

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 with a target position. Therefore, the movement has a defined end.
Continuous motion	The drive is executing a movement without a target position. The movement has no defined end.
Synchronized motion	The drive is coupled to another drive.

Table: Overview of the drive states

Transitions between these states are initiated by calling certain commands.

This can result in the following sequence for example:

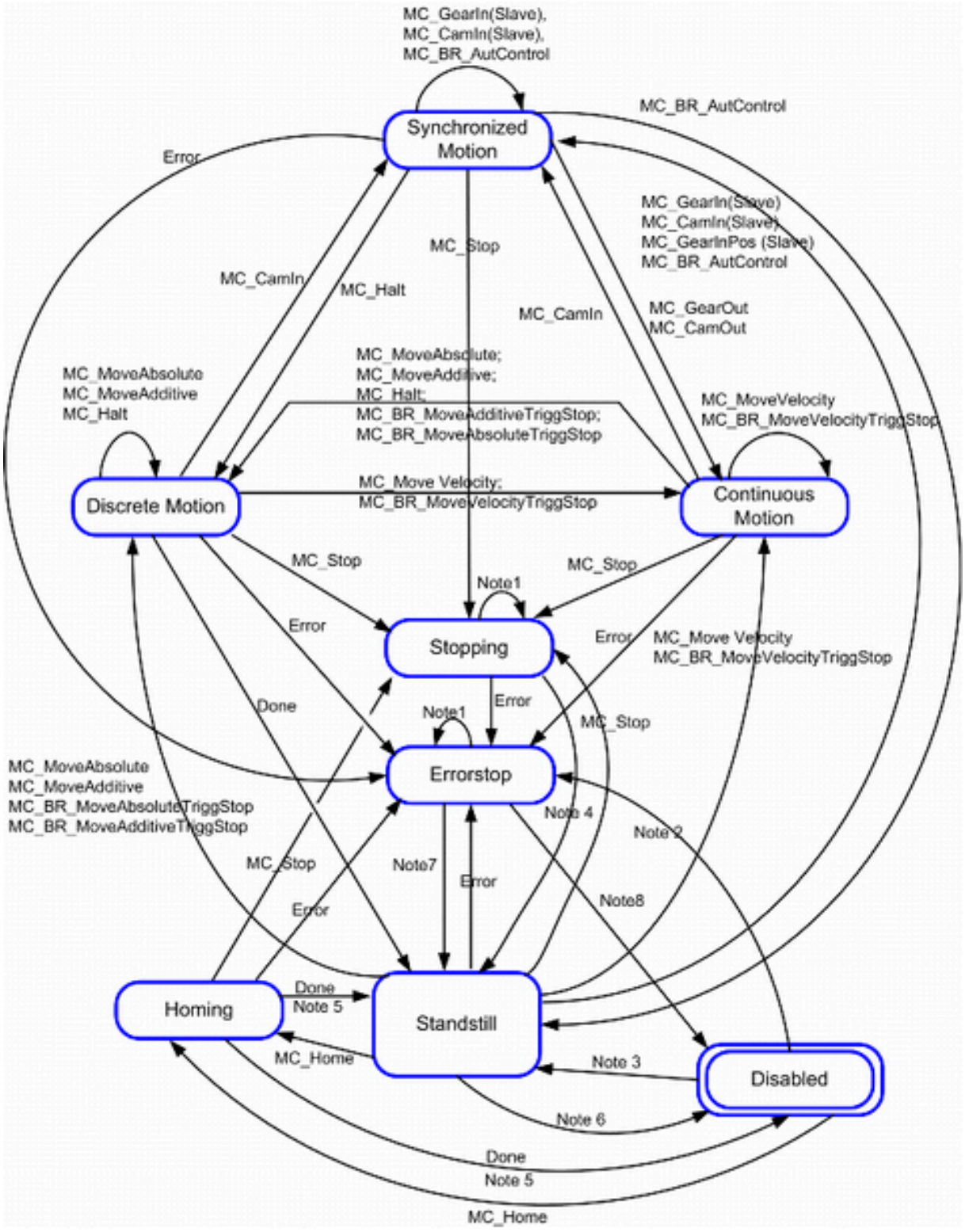


Progression of states

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

After the target position has been reached, the drive returns to its "initial state". If a drive error occurs during the positioning action, then the axis enters error state (Errorstop). This can be acknowledged using the MC_Reset function after the drive error has been corrected.

Complete overview of all states:



PLCopen Motion Control Diagram of States

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

The most important drive states are included in the diagram. They can be used for coordinating positioning sequences. The MC_ReadStatus function block is used to read the current status from an axis.

A virtual axis does not have a drive controller. Unlike the real axis, the virtual axis does not have a Disabled status. The NC object "virtual axis" starts in the Standstill state and does not need the MC_Power function block for activation.

Exercise: State monitoring

The current drive state can be determined using the **MC_ReadStatus** function block. Add this function to your test program and monitor the state changes when executing different positioning actions.

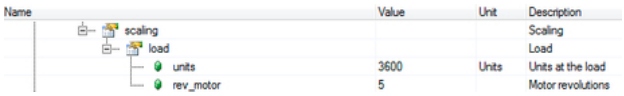
5.3 Periodic axes - Rotary axis

Additional settings allow us to make adjustments to position values.

Periodic position behavior as well as position scaling can be achieved using these simple configurations. This can be used in turntable applications, for example.

What do we need to do first?

The basic settings for scaling a revolution in units are made using the encoder interface parameters for the axis:

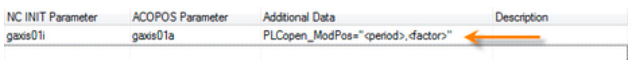


Init parameter module, encoder interface

In the image above, 3600 units are divided into five axis revolutions, resulting in 720 units per revolution. This makes it possible to divide the revolutions according to requirements (keeping the maximum resolution of the encoder in mind). DINT is the data type for the position.

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

Values for the position period and position factor can be specified for the PLCopen_ModPos="<Period>,<Factor>" entry in the NC mapping table to adjust the position value:

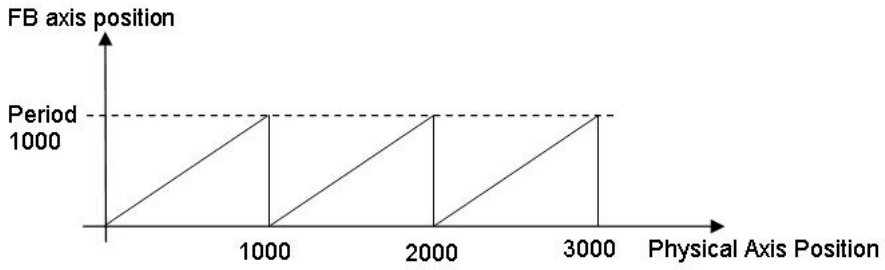


NC mapping table, advanced settings

Axis period

For continuous axis movements, it is frequently necessary to determine a position value periodically. If a value greater than 0 is being used for the period, then the position value is adjusted according to this entry. It refers directly to the scaling of the axis in the encoder parameters. All PLCopen function blocks "work" with this periodic position.

For example, if the value <Period> = 1000, then the position value always increases from 0 to 999 during a positive movement before being reset to 0 and increasing again to 999.



Periodic axis position

Periodic position conversion can be disabled by entering <Period>=0 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 ($\pm 2^{23}$) because this is the largest number which can be displayed by the REAL data type without losing accuracy.

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

Axis factor

PLCopen function blocks use the REAL data type for the axis position. This data type allows us to use decimal places, which in turn can make simplified scaling pretty interesting.

What exactly is "scaling"?

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. Let's assume that the way we've configured the encoder parameters allows us to set up a 1 µm distance per positioning unit.

Now it might be beneficial if we could specify the distance in millimeters for our positioning task with PLCopen function blocks. And that is precisely what is achieved with this factor. The equation looks like this: $PLCopenUnits = \frac{AxisparameterUnits}{Factor}$

So if we set the factor to a value of 1000, we will get our scaling to millimeters. If we now start a movement for a distance of value 1 with the corresponding PLCopen function block, our axis will actually travel 1000 axis parameter units.

<Factor>	PLCopen units [REAL]	Axis parameter units [DINT]
1	1	1
1000	1	1000
1000	0001	1

Table: Comparing PLCopen units to axis units according to the set factor

Velocity and acceleration values also refer to this scaling.

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

Exercise: Configuring a rotary axis

This simple example will implement the settings shown above and demonstrate the resulting possibilities.

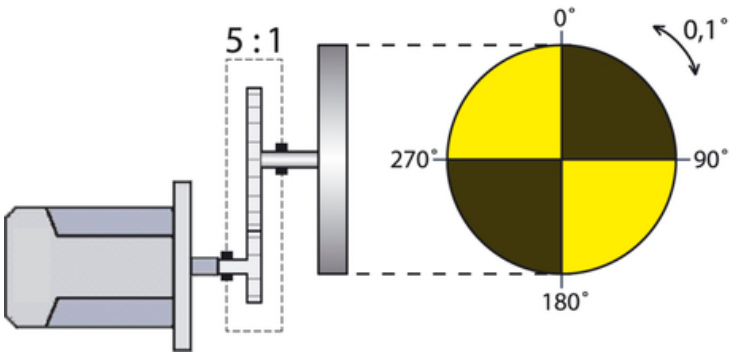
Task definition:

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

Positioning must be within 0.1° of accuracy. The MC_MoveAbsolute function block will be used for approaching the positions.

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

```
MC_MoveAbsolute_0.Position:= 135.0; (* goal: 135° *)
```



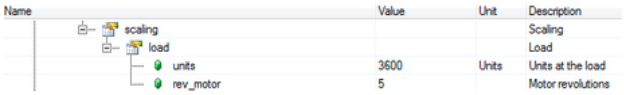
Sketch of the mechanical structure

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

Find the suitable encoder settings and advanced position value settings for this task.

Possible solution:

The basic settings for the encoder are made in the Init parameter module. Therefore, 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.



Init parameter module, encoder interface, configuration

If a movement of 3600 units is now executed, then the motor performs exactly 5 revolutions and the carrier is rotated exactly 360°. The gear has now been fully configured.

To now get the desired scaling for the positioning function, the value 10 must be defined for the factor and the value 3600 for the period (based directly on the encoder setting):

Advanced settings for the periodic axis in the mapping table

6 TIPS FOR PROGRAMMING

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

For a structured sequence in the application program, a state machine (step sequencer) can be used for high-level languages.

A detailed look at the application program

6.1 Uses of control structures

Each function block can be executed as needed (Enable, Execute). Processes can be started at a defined point in the program using commands.

Status outputs and output parameters provide information about the current state:

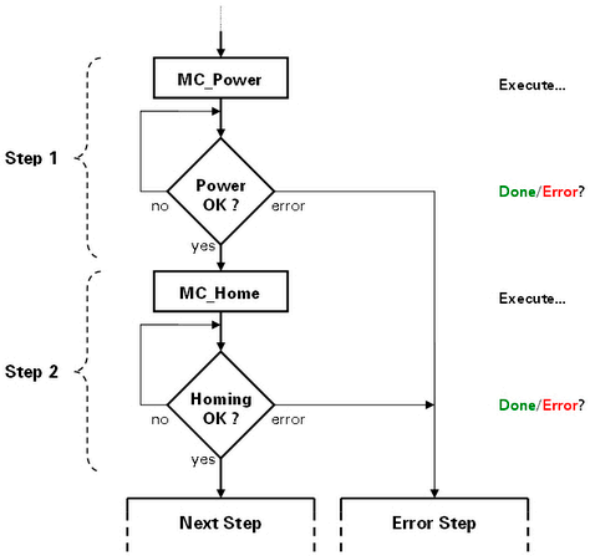
- Execution successful?
- If not, which error occurred?
- Status of the process:
  - Is the axis moving?
  - Has the axis arrived at the target position?
  - Was the homing procedure successful?

This information can be used to control subsequent steps in the program sequence. If an error occurs, the program has to respond differently depending on the error that occurs.

The step sequencer is a control structure that is especially well-suited for managing these types of function sequences.

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

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



Sample control structure, structured programming

The function blocks can be activated with Execute in the individual steps of the control structure. Status parameters such as Error, Done and ErrorID can be used to determine the step where the application will continue.

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

Samples using this concept that have already been implemented can be imported in Automation Studio (see 7 "Using the sample programs").

6.2 Error handling in the application program

When programming an application, it is important to always take error evaluation in the program sequence into consideration. The main question is, which errors must be considered?

There are two types of errors:

- Errors when calling a function block
- Drive errors

Evaluating errors in the application program

Programming \ Libraries \ Motion libraries \ ACP10_MC \ Function blocks \ Error numbers

Motion \ Reference manual \ ACP10 \ ACOPOS error texts

How can these errors be monitored?

PLCopen function blocks provide direct feedback about their status. An error that occurs during execution is displayed on the Error status output. When this happens, a corresponding error number is output on the function block's ErrorID output to more precisely localize the error.

The MC_ReadAxisError function block is available to monitor drive errors.

If an error occurs when calling a function block, it's absolutely necessary to reset the block before the next time it is used so that the error status remains. This is done by executing the function block with Enable=0 or Execute=0.

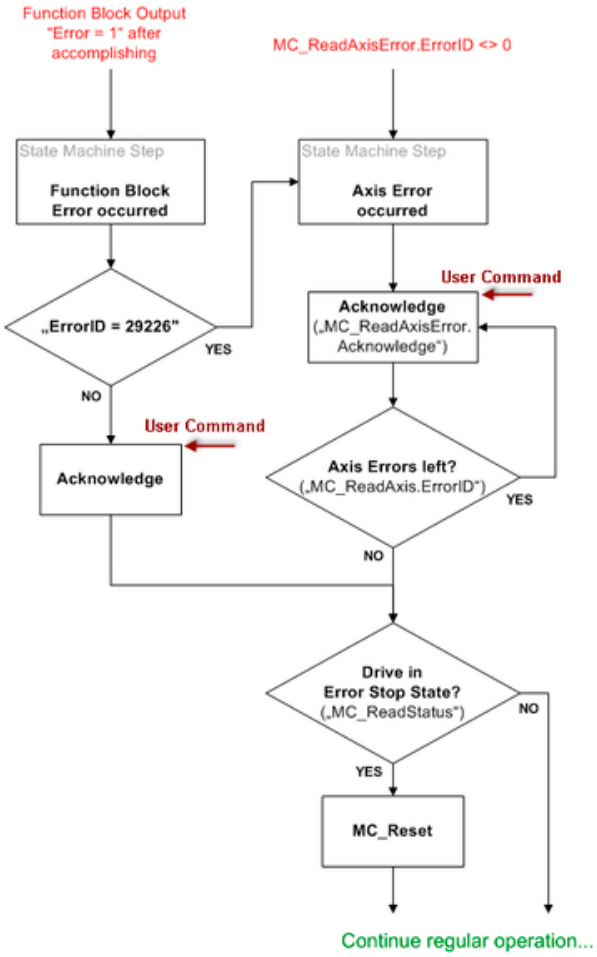
One possible approach to error evaluation:

We recommend using the MC_ReadAxisError function block cyclically in the program to monitor drive errors.

The function block must be constantly active for this (Enable = 1). The necessary query (error?) can be placed in a cyclic program section, e.g. before the control structure.

When a drive error is registered, then regular program execution should be stopped and a step for handling the error should be entered.

The same is true when an error occurs during the actual function block call. There should also be an "error step" implemented to handle this situation.



Procedure for error handling in a positioning application

Additionally, the **MC_ReadAxisError** (**MC_BR_ReadAxisError**, **MC_BR_AxisErrorCollector**) function block offers the possibility to output plain text concerning the current drive error using a STRING variable.

In order to use this option, the "NC error text table" object needs to be managed in the desired language of the project.

The name of this module must be connected to the function block. Once this is done, the text for the current error number is then sent from the function block to the specified STRING variable.

We recommend running error evaluation before the step sequencer that controls the axis. This allows axis errors to be read by the application immediately and not one cycle later.

Error evaluation can take place in a series of steps:

- Drive errors can be checked and acknowledged sequentially. The function block returns the drive errors one after the other until the last one has been acknowledged.

- With PLCopen function block errors, the system first checks to determine if the error was caused by the drive (ErrorID 29226). It is then possible to acknowledge the errors and exit the error step.
- After both routines have been executed, the drive status should be checked as well. In certain cases, the drive can be set to the "Error stop" status. This status will remain in effect until a status reset (MC_Reset) is carried out (see 5.2 "Overview of the drive states").

The application can then be continued once all errors have been corrected and acknowledged.

This program sequence should in no way be seen as a guideline for implementing error handling. Instead, it should merely serve as a universal approach or basis for implementing this type of routine. Detailed information can be found in the sample programs (see 7 "Using the sample programs").

In addition, the sample program for multi-axis operation included with the Automation Studio installation provides valuable ideas for implementing error handling ("ErrorHandling" program).

Programming \ Examples \ Motion \ Motion control \ Multi-axis operation

7 USING THE SAMPLE PROGRAMS

The Automation Studio installation includes sample programs for positioning applications that use the most important functions in the ACP10_MC library. This makes it possible to add a functional application section to a project that already includes axes.

Basic movement types can thus be handled in the control application without additional programming.

Programming \ Examples \ Motion \ Motion Control

7.1 Overview - Adapting samples

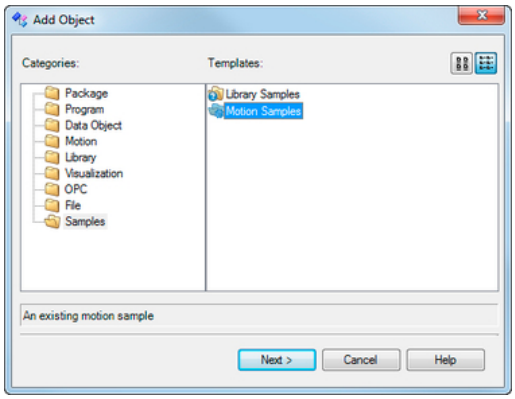
The Automation Studio installation also includes several sample programs that can be used as a basis for fast and easy drive control in the control application.

The samples are provided in the Ladder Diagram, Structured Text and ANSI C programming languages.

Samples for single axes, axis linking, cam profiles and other applications are available.

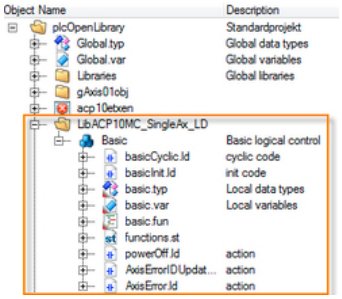
Importing samples

The samples that are available can be added to an Automation Studio project in the logical view using wizards. In the wizards, there is a **Samples** field with the subgroup **Motion samples**.



Adding a motion sample to the Automation Studio project

All required components are then added to the Automation Studio project, and the sample program is stored in a separate package in the logical view.



Program sample stored in a separate package

A precise overview of these samples and their names can be found in the Automation Studio help documentation.

Programming \ Examples \ Motion \ Motion Control

Sample functionality

Samples are configured according to the recommendations for programming discussed previously (see 6 "Tips for programming".

In a large structure, there is a substructure for commands, axis parameters and status messages that can be used to operate the machine. The axis reference can be modified in the sample's Init subprogram.

A description of necessary steps and how to design the control structures can be found for each sample in the Automation Studio help documentation.

Exercise: Using a sample program

Insert the sample program for a basis movement in the Structured Text programming language.

1) Check the description in the AS help documentation.
2) Import the sample in the logical view.
3) Modify the axis reference in the Init subprogram for the sample program as needed.
4) Upload the application.
5) Restart the CPU.
6) Test the commands and check the results of the sample program.

8 MANAGING DRIVE PARAMETERS

A few function blocks are provided in the ACP10_MC library for managing the parameters on the drive.

These parameters can be used to set up the drive as well as to handle positioning tasks. In general, these parameters are managed by the NC operating system.

In some applications, however, direct access (read and write) to different parameters is required during runtime.

By manipulating ParIDs as needed, the drive configuration can be adapted to handle specific situations.

These functions are only offered for the B&R drive solution, i.e. they can only be implemented using B&R-specific function blocks. The use of these function blocks (operation, status check, etc.) still conforms to the PLCopen Motion Control standard, however.

Motion / Reference manual / ACP10 / ACOPOS parameter IDs

8.1 Initializing and reading individual parameters

Individual parameters can be initialized and read a single time or cyclically.

Function blocks for initializing or reading a single time:

- MC_BR_WriteParID
- MC_BR_ReadParID

Individual ParIDs can be initialized or read automatically with each TC#1 cycle by setting up a cyclic transfer:

- MC_BR_InitCyclicWrite
- MC_BR_InitCyclicRead
- MC_BR_CyclicWrite
- MC_BR_CyclicRead

Initializing means making a specific value "operational" for a parameter on the drive. Initialization does not always take play automatically after a transfer. Some functions only transfer parameter values to the drive. In these cases, initialization can be carried out at a later point.

Function blocks for communicating between POWERLINK nodes:

- MC_BR_InitSendParID
- MC_BR_InitReceiveParID
- MC_BR_ReceiveParIDOnPLC

An additional function block can be used to establish independent cyclic communication for a ParID between individual ACOPOS devices:

- MC_BR_InitMasterParIDTransfer

This function block is used for a cam profile automat's additive axes and for Smart Process Technology functions.

Programming \ Libraries \ Motion libraries \ ACP10_MC \ Function blocks \ Administration of ACOPOS ParIDs

8.2 Transferring and initializing parameter sets

Other function blocks for managing drive parameters support the transfer and initialization of multiple ParIDs.

There are three ways to do this:

- Parameter tables
- Parameter lists
- Parameter sequences

Parameter tables

Parameters entered in an ACOPOS parameter table are individually transferred to the drive in each NC Manager task cycle and immediately initialized using the **MC_BR_InitParTableObj** function block.

Name	ID	Value	Unit	Description
Parameters				
CMD_SIMULATION	110	ncSW_ON		Simulation mode: Command
MOTOR_TEST_MODE	866	7		Motor: Test mode
PHASE_MON_IGNORE	80	1		Power mains: Ignore phase failure
UDC_NOMINAL	390	24	V	CTRL DC bus: Nominal voltage

New ACOPOS parameter table

Parameter lists

Parameter configuration and value assignment can be easily adjusted during runtime using a parameter list.

There is an additional data type for this purpose that contains an element for the parameter ID and an element for the parameter value. You can use this data type to create an array in the program. Entries in this list can then be changed from inside the application program during runtime.

All parameters in the parameter list, like ACOPOS parameter tables, are transferred to the drive in each NC Manager task cycle and immediately initialized using the **MC_BR_InitParList** function block.

Unlike the initialization variant using predefined ACOPOS parameter tables, parameter arrays make it possible to change both the parameters contained inside the array as well as their values while the application is running.

Parameter sequences

A parameter sequence uses the same array as the parameter list.

Unlike ACOPOS parameter tables or parameter lists, however, the parameters in a parameter sequence are transferred as a data block (not individually) to the drive during the NC Manager idle time task (**MC_BR_DownloadParSequ**). Once there, it is not immediately initialized; instead, it is only saved as a "recipe".

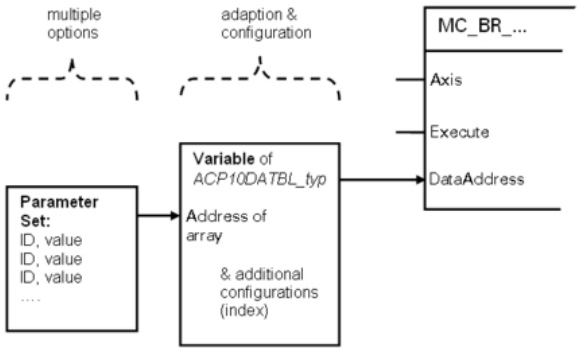
These parameters can then be activated with the "Initialize sequence" command (**MC_BR_InitParSequ**).

Because each parameter is assigned an index, it is possible to place multiple parameter configurations together on the drive and initialize them selectively.

Programming \ Libraries \ Motion libraries \ ACP10_MC \ Function blocks \ Administration of ACOPOS ParIDs

8.2.1 Additional information

Each of these function blocks includes a "DataAddress" input parameter for connecting parameters grouped together in a parameter list or parameter sequence. An ACP10DATBL_typ structure variable still must be provided here as an "intermediary".



Connecting a parameter group via an ACP10_DATBL_typ variable

The diagram shows a simplified version of this relationship. The parameter group is located on the left as a parameter array. The group is connected to the function block via the variable for transfer configuration (ACP10DATBL_typ). The parameters can be downloaded once the assignments have been made correctly ("Execute= 1").

8.3 Reading and writing parameters cyclically

Depending on the application, additional drive information or changes to the closed loop drive control may be required in the control program. Typically, this is done by cyclically reading and writing parameters. Possibilities were already illustrated in one of the previous sections (see 8.1 "Initializing and reading individual parameters").

However, there are not only functional blocks for reading and writing parameter IDs, but also function blocks that relate directly to functionality.

Exercise: Expanding the sample program - Reading

Expand the sample program to include the following functions:

- Reading the motor current
- Reading the current motor temperature
- Reading the current torque

ParID	Description
214	Actual stator current of the quadrature component
381	Temperature sensor: Temperature
277	Motor: Torque

Table: Overview of the drive parameters to be read

Exercise: Expanding the sample program - Writing

Expand the sample program to include the following functions:

- Limiting torque with MC_BR_WriteParID
- Creating an array of USINT [1..4] for the description of the override group (ncPAR_TYP_VOID)

ParID	Description
343	CTRL torque limiter: Override

Table: Overview of the drive parameters to be read

Because there are a total of four parameters to be written, the data should be transferred in an array of data type ncPAR_TYP_VOID.

Defining cyclic set values

Some applications require that the set value be calculated on the controller. The axis that should follow this set value must be supplied with this data cyclically.

The following function blocks are available for this purpose:

- MC_BR_MoveCyclicPosition
- MC_BR_MoveCyclicVelocity
- MC_BR_VelocityControl

Programming \ Libraries \ Motion libraries \ ACP10_MC \ Function blocks \ Cyclic set values

9 CONTROLLING SEVERAL SINGLE AXES - SOFTWARE CONFIGURATION

In a control project, it is often necessary to operate several single axes. Although it is possible to create a separate program for each drive axis in the logical view, this is not very efficient. For example, software changes must be performed multiple times, which makes use and maintenance extremely prone to errors.

The following section explains some suggested solutions for controlling several single axes using a single program.

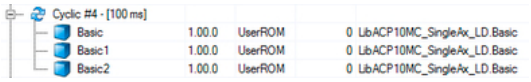
A program with a loop

The application program consists of a structure that contains a substructure for commands, parameters, states and function block instances for PLCopen functions.

In the program, the axes are operated and the function blocks are called within a loop.

Identification of the program names in the Init code

In Automation Studio, it is possible to map programs more than once with respect to the software configuration. Here, the program for controlling the drive is linked to the cyclic system multiple times.



Mapping a program multiple times in the software configuration

The program is compiled and executed multiple times on the controller. In the application, the axis reference must be assigned for each instance of the program. Here, it is necessary to determine which of the tasks that are assigned to the software configuration is being referred to, e.g. in the Init subprogram.

This can be determined using the functions in the "SYS_lib" library. A link is then established between the program code and the task names.

Declaration

```
VAR
    sTaskName :  STRING[80];
    status :  UINT;
    AxRef :  UDINT;
END_VAR
```

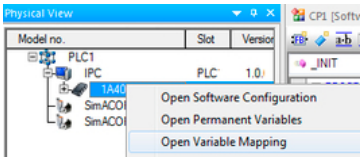
Program code:

```
(*Determining the particular task names*)
status := ST_name(0, ADR(sTaskName), 0);

(* Program name Axis1? *)
IF sTaskName = 'Axis1' THEN
    AxRef :=  ADR(gAxis01);
ELSE
    AxRef :=  ADR(gAxis02);
END_IF
```

Table: Determining the particular task names in structured text with "ST_name"

The local command variable can then be linked with a higher-level management program that controls the individual axes using variable assignment. The variable assignment window can be opened in the physical view using the CPU's shortcut menu.



Opening the variable assignment window using the CPU's shortcut menu

Furthermore, it is also possible for the management program to transparently access and control the command structure of the single axis program using a pointer.

Programming \ Libraries \ Configuration, system information, runtime control \ SYS_lib \ Function blocks \ Handling software objects

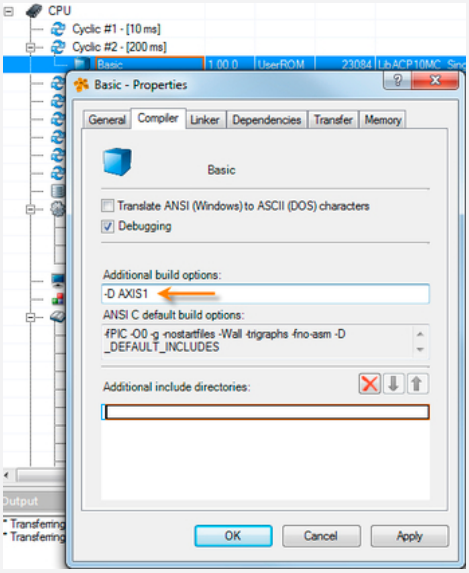
Programming \ Editors \ Configuration editors \ Assigning variables

Using preprocessor directives

This method only differs from the "Identification of the program names in the Init code" variant in that the preprocessor is used instead of the functions in the SYS_lib library to determine which axis reference is assigned to the single axis program.

Here, an identifier is specified in the compiler options for the task in the software configuration which can be accessed in the program when compiling.

In the software configuration, a "Define" is specified in the task properties.



Specifying a "Define" with the option "-D AXIS1" in the task properties of the software configuration

In the program code, it is now possible to query which "Defines" are specified.

Program code:

```
#ifdef AXIS1
    AxRef:= ADR(gAxis01);
#else
    AxRef:= ADR(gAxis02);
#endif
```

Table: Querying the "Defines" in the Init subprogram

Project management \ The workspace \ General project settings \ Settings for IEC compliance

Programming \ Programs \ Preprocessor for IEC programs

10 SUMMARY

Powerful function blocks are provided for controlling the B&R drive solution. These are created based on the PLCopen Motion Control standard and feature a uniform design regarding functional usage.

Once the user is familiar with this standardized operation, it is possible to begin combining the corresponding function blocks needed for the process from the pool of functions included in the Motion Control library (ACP10_MC).

The automatic sequence can be optimally implemented using a step sequencer. Error handling routines turn the sequence into a complete positioning application.

PLCopen Motion Control logo

In addition to the standard, specific function blocks are also provided for controlling drives. They handle special functions relating to the B&R drive solution. This enables the programmer to access the full range of functions to solve any positioning task.

TRAINING MODULES

TM210 – Working with Automation Studio
TM213 – Automation Runtime
TM220 – The Service Technician on the Job
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
TM261 – Closed Loop Control with LOOPCONR
TM400 – Introduction to Motion Control
TM410 – Working with Integrated Motion Control
TM440 – Motion Control: Basic Functions
TM441 – Motion Control: Multi-axis Functions
TM450 – ACOPOS Control Concept and Adjustment
TM460 – Initial Commissioning of Motors
TM480 – The Basics of Hydraulics
TM481 – Valve-based Hydraulic Drives
TM482 – Hydraulic Servo Pump Drives
TM500 – Introduction to Integrated Safety
TM510 – Working with SafeDESIGNER
TM530 – Developing Safety Applications
TM540 – Integrated Safe Motion Control
TM600 – Introduction to Visualization
TM610 – Working with Integrated Visualization
TM630 – Visualization Programming Guide
TM640 – Alarms, Trends and Diagnostics
TM670 – Advanced Visual Components
TM810 – APROL Setup, Configuration and Recovery
TM811 – APROL Runtime System
TM812 – APROL Operator Management
TM813 – APROL XML Queries and Audit Trail
TM830 – APROL Project Engineering
TM890 – The Basics of LINUX

www.br-automation.com

TM440TRE.00-ENG / V1.0.3