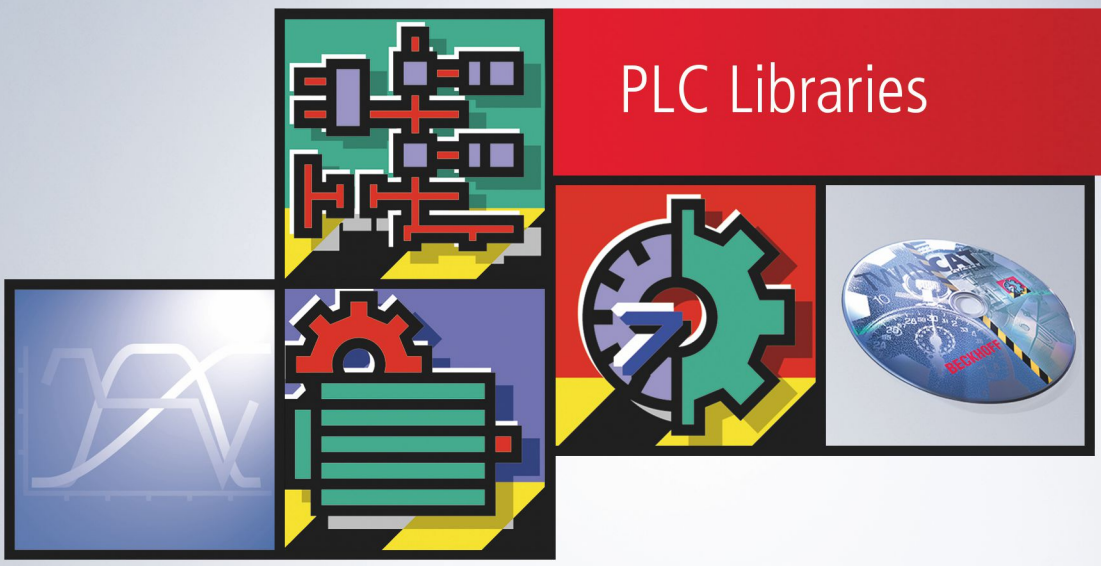
BECKHOFF New Automation Technology

Manual | EN

TX1200

TwinCAT 2 | PLC Library: TcMC2



2023-07-20 | Version: 1.3

Table of contents

1 Foreword 7
1.1 Notes on the documentation 7
1.2 For your safety 8
1.3 Notes on information security 9
2 Overview 10
3 State diagram 11
4 General rules for MC function blocks 14
5 Migration from TcMC to TcMC2 17
6 Organisation function blocks 19
6.1 Axis functions 19
6.1.1 MC_Power 19
6.1.2 MC_Reset 20
6.1.3 MC_SetPosition 21
6.2 Status and parameter 23
6.2.1 MC_ReadActualVelocity 23
6.2.2 MC_ReadActualPosition 24
6.2.3 MC_ReadAxisComponents 24
6.2.4 MC_ReadAxisError 25
6.2.5 MC_ReadBoolParameter 26
6.2.6 MC_ReadParameter 27
6.2.7 MC_ReadParameterSet 29
6.2.8 MC_ReadStatus 30
6.2.9 MC_WriteBoolParameter 32
6.2.10 MC_WriteParameter 33
6.3 Touch probe 34
6.3.1 MC_TouchProbe 34
6.3.2 MC_TouchProbe_V2 37
6.3.3 MC_AbortTrigger 40
6.3.4 MC_AbortTrigger_V2 41
6.4 External set value generator 42
6.4.1 MC_ExtSetPointGenEnable 42
6.4.2 MC_ExtSetPointGenDisable 43
6.4.3 MC_ExtSetPointGenFeed 44
6.5 Special extensions 45
6.5.1 MC_PowerStepper 45
6.5.2 Notes on the MC_PowerStepper 46
6.5.3 MC_OverrideFilter 51
6.5.4 MC_SetOverride 52
6.5.5 MC_SetEncoderScalingFactor 53
6.5.6 MC_PositionCorrectionLimiter 54
6.5.7 MC_ReadDriveAddress 55
6.5.8 MC_SetAcceptBlockedDriveSignal 56
7 Motion function blocks 58

7.1 Point to point motion ... 58
7.1.1 MC_MoveAbsolute ... 58
7.1.2 MC_MoveRelative ... 60
7.1.3 MC_MoveAdditive ... 62
7.1.4 MC_MoveModulo ... 64
7.1.5 Notes on modulo positioning ... 66
7.1.6 MC_MoveVelocity ... 72
7.1.7 MC_MoveContinuousAbsolute ... 74
7.1.8 MC_MoveContinuousRelative ... 76
7.1.9 MC_Halt ... 78
7.1.10 MC_Stop ... 80
7.2 Superposition ... 82
7.2.1 MC_MoveSuperimposed ... 82
7.2.2 Application examples for MC_MoveSuperimposed ... 84
7.2.3 MC_AbortSuperposition ... 87
7.3 Homing ... 88
7.3.1 MC_Home ... 88
7.4 Manual motion ... 91
7.4.1 MC_Jog ... 91
7.5 Axis coupling ... 93
7.5.1 MC_GearIn ... 93
7.5.2 MC_GearInDyn ... 95
7.5.3 MC_GearOut ... 97
7.5.4 MC_GearInMultiMaster ... 98

8 Data types ... 101
8.1 Axis interface ... 101
8.1.1 Data type AXIS_REF ... 101
8.1.2 Data type NCTOPLC_AXIS_REF ... 102
8.1.3 Data type PLCTONC_AXIS_REF ... 102
8.2 Motion function blocks ... 103
8.2.1 Data type MC_BufferMode ... 103
8.2.2 Data type MC_Direction ... 105
8.2.3 Data type MC_HomingMode ... 106
8.2.4 Data type E_SuperpositionMode ... 106
8.2.5 Data typeST_SuperpositionOptions ... 107
8.2.6 Data type E_JogMode ... 108
8.3 Status and parameter ... 108
8.3.1 Data type E_ReadMode ... 108
8.3.2 Data type ST_AxisStatus ... 109
8.3.3 Data type MC_AxisParameter ... 110
8.3.4 Data type ST_PowerStepperStruct ... 111
8.3.5 Data type ST_DriveAddress ... 111
8.3.6 Data type ST_AxisParameterSet ... 111
8.3.7 Data type ST_AxisOpModes ... 113
8.3.8 Data type E_AxisPositionCorrectionMode ... 113
8.3.9 Data type MC_AxisStates ... 113

8.4 Touch probe 114
8.4.1 Data type TRIGGER_REF 114
8.4.2 Data type MC_TouchProbeRecordedData 115
8.5 External set value generator 115
8.5.1 Datentyp E_PositionType 115
9 Example programs 116
9.1 Sample Programs 116

1 Foreword

1.1 Notes on the documentation

This description is only intended for the use of trained specialists in control and automation engineering who are familiar with applicable national standards.
It is essential that the documentation and the following notes and explanations are followed when installing and commissioning the components.
It is the duty of the technical personnel to use the documentation published at the respective time of each installation and commissioning.

The responsible staff must ensure that the application or use of the products described satisfy all the requirements for safety, including all the relevant laws, regulations, guidelines and standards.

Disclaimer

The documentation has been prepared with care. The products described are, however, constantly under development.
We reserve the right to revise and change the documentation at any time and without prior announcement.
No claims for the modification of products that have already been supplied may be made on the basis of the data, diagrams and descriptions in this documentation.

Trademarks

Beckhoff®, TwinCAT®, TwinCAT/BSD®, TC/BSD®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® and XPlanar® are registered trademarks of and licensed by Beckhoff Automation GmbH.
Other designations used in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owners.

Patent Pending

The EtherCAT Technology is covered, including but not limited to the following patent applications and patents:
EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
with corresponding applications or registrations in various other countries.

EtherCAT® is a registered trademark and patented technology, licensed by Beckhoff Automation GmbH, Germany

Copyright

1.2 For your safety

Safety regulations

Read the following explanations for your safety.
Always observe and follow product-specific safety instructions, which you may find at the appropriate places in this document.

Exclusion of liability

All the components are supplied in particular hardware and software configurations which are appropriate for the application. Modifications to hardware or software configurations other than those described in the documentation are not permitted, and nullify the liability of Beckhoff Automation GmbH & Co. KG.

Personnel qualification

This description is only intended for trained specialists in control, automation, and drive technology who are familiar with the applicable national standards.

Signal words

The signal words used in the documentation are classified below. In order to prevent injury and damage to persons and property, read and follow the safety and warning notices.

Personal injury warnings

⚠ DANGER
Hazard with high risk of death or serious injury.

⚠ WARNING
Hazard with medium risk of death or serious injury.

⚠ CAUTION
There is a low-risk hazard that could result in medium or minor injury.

Warning of damage to property or environment

NOTICE
The environment, equipment, or data may be damaged.

Information on handling the product

This information includes, for example:
recommendations for action, assistance or further information on the product.

1.3 Notes on information security

The products of Beckhoff Automation GmbH & Co. KG (Beckhoff), insofar as they can be accessed online, are equipped with security functions that support the secure operation of plants, systems, machines and networks. Despite the security functions, the creation, implementation and constant updating of a holistic security concept for the operation are necessary to protect the respective plant, system, machine and networks against cyber threats. The products sold by Beckhoff are only part of the overall security concept. The customer is responsible for preventing unauthorized access by third parties to its equipment, systems, machines and networks. The latter should be connected to the corporate network or the Internet only if appropriate protective measures have been set up.

In addition, the recommendations from Beckhoff regarding appropriate protective measures should be observed. Further information regarding information security and industrial security can be found in our https://www.beckhoff.com/secguide.

Beckhoff products and solutions undergo continuous further development. This also applies to security functions. In light of this continuous further development, Beckhoff expressly recommends that the products are kept up to date at all times and that updates are installed for the products once they have been made available. Using outdated or unsupported product versions can increase the risk of cyber threats.

To stay informed about information security for Beckhoff products, subscribe to the RSS feed at https://www.beckhoff.com/secinfo.

2 Overview

The TcMC2 TwinCAT motion control PLC library includes function blocks for programming machine applications and represents a further development of the TcMC library. TcMC2 is based on the revised PLCopen specification for motion control function blocks V2.0 (www.PLCopen.org).

Compatibility

The TcMC2 motion control library contains enhanced and new functions. The function blocks are better adapted to the requirements of the PLCopen specification and are not compatible with the first version (TCMC). Users who maintain existing projects are recommended to continue working in these projects with the classic TcMC. TcMC2 should be used for new projects or for the revision of existing projects.

Main new features

A key feature of TcMC2 compared with TcMC is the so-called buffer mode. Buffer mode enables Move commands to be queued in order to achieve a continuous positioning without intermediate stops. It enables transition of two travel commands with a defined velocity at a certain position.

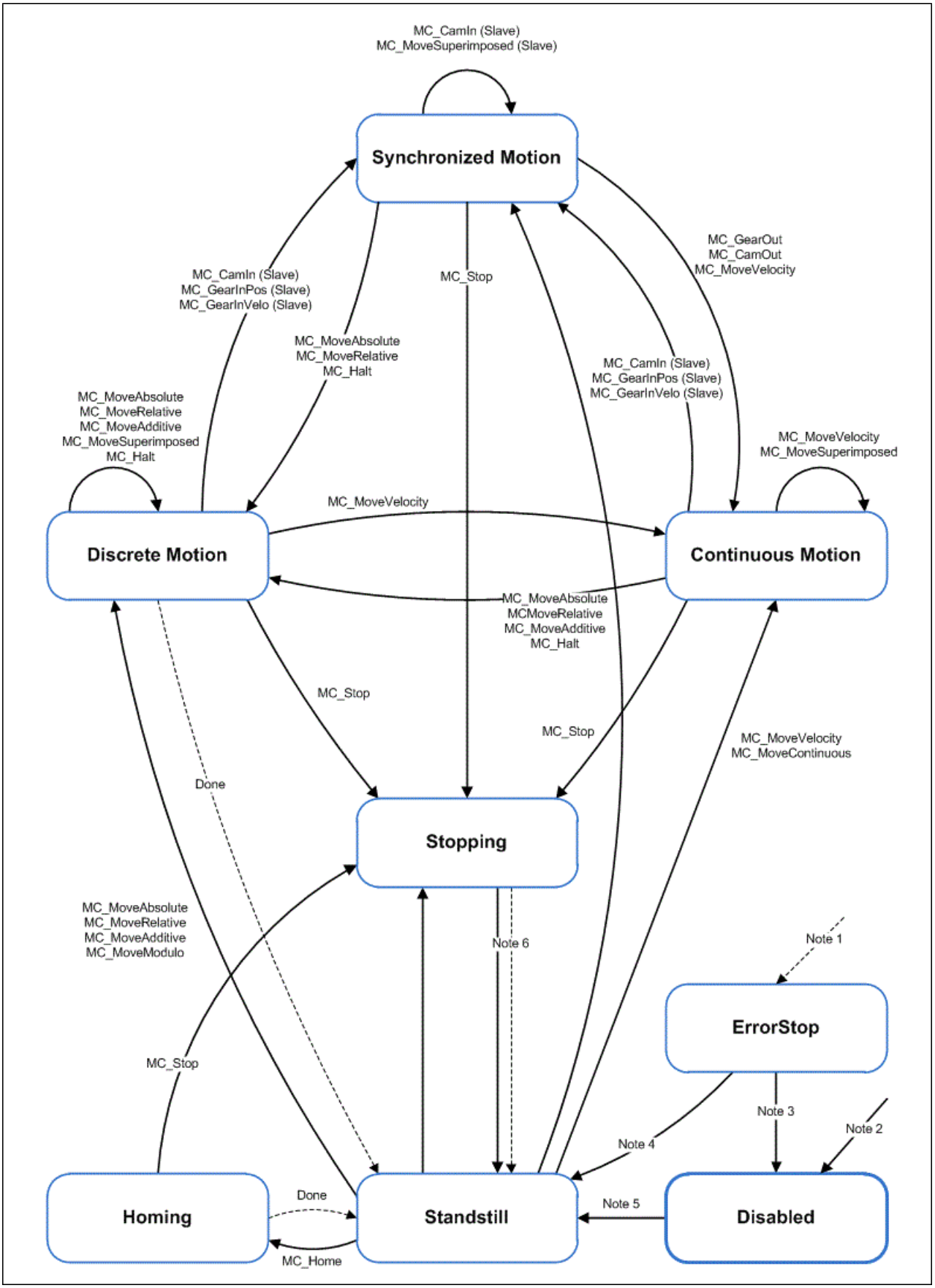
Move commands can be followed by further Move commands during execution. This makes adaptation of target position or travel speed during the movement much easier.

TwinCAT Version

The TcMC2 library can be used with TwinCAT version 2.10 Build 1340 or higher. With remote programmed controllers care must be taken that an appropriate version is installed on both the programmer PC and the control PC. In the case of control systems with the operating system Windows CE, the version of the installed image is decisive. A Windows CE image with version 3.08 or higher is required here.

3 State diagram

The following state diagram defines the behavior of an axis in situations where several function blocks are simultaneously active for this axis. The combination of several function blocks is useful for generating more complex motion profiles or for dealing with exceptional situations during program execution.



Note 1	From any state in which an error occurs
Note 2	From any state if *MC_Power.Enable* = FALSE and the axis has no error

Note 3	MC_Reset and *MC_Power.*Status = FALSE
Note 4	MC_Reset and *MC_Power.*Status = TRUE and *MC_Power*.Enable = TRUE
Note 5	MC_Power.Status = TRUE and *MC_Power*.Enable = TRUE
Note 6	MC_Stop.Done= TRUE and *MC_Stop.Execute* = FALSE

As a basic rule, travel commands are processed sequentially. All commands operate within this axis state diagram.

The axis is always in one of the defined states. Motion commands resulting in a transition change the axis state and, as a result, the motion profile. The state diagram is an abstraction layer that reflects the real axis state, comparable to the process image for I/O points. The axis state changes immediately when the command is issued.

The state diagram initially targets single axes. Multi-axis blocks such as *MC_CamIn* or *MC_GearIn* influence the states of several axes, which can always be traced back to individual axis states of the axes involved in the process. For example, a cam plate master can be in *Continous Motion* state, while the associated slave is in *Synchronized Motion* state. Coupling of a slave has no influence on the state of the master.

The *Disabled* state is the default state of an axis. In this state can the axis cannot be moved through a function block. If the MC Power [▶ 19] block is called with Enable=TRUE, the axis changes to state *Standstill* or, on error, *ErrorStop*. If *MC_Power* is called with Enable=FALSE, the state changes to *Disabled*

The purpose of status *ErrorStop* is to stop the axis and then block further commands, until a reset was triggered. The Error state transition only refers to actual axis errors, not function block execution errors. Axis errors may also be indicated at the error output of a function block.

Function blocks that are not listed in the state diagram have no influence on the axis state. (*MC_ReadStatus; MC_ReadAxisError; MC_ReadParameter; MC_ReadBoolParameter; MC_WriteParameter; MC_WriteBoolParameter; MC_ReadActualPosition* and *MC_CamTableSelect.*)

The *Stopping* state indicates that the axis is in a stop ramp. Once the axis has stopped the state changes to *StandStill*.

Travel commands such as *MC_MoveAbsolute* that lead out of the *Synchronized Motion* state are possible only if they are explicitly permitted in the axis parameters. Uncoupling commands such as *MC_GearOut* are possible independent of that.

4 General rules for MC function blocks

For all MC function blocks the following rules apply, which ensure defined processing through the PLC program.

Exclusivity of the outputs

The outputs *Busy, Done, Error* and *CommandAborted* are mutually exclusive, i.e. only one of these outputs can be TRUE at a function block at any one time. When the *Execute* input becomes TRUE, one of the outputs must become TRUE. Similarly, only one of the outputs *Active, Error, Done* and *CommandAborted* can be TRUE at any one time.

An exception to this rule is MC_Stop [▶ 80]. *MC_Stop* sets *Done* to TRUE as soon as the axis is stopped. Nevertheless, *Busy* and *Active* remain TRUE because the axis is locked. The axis is unlocked and *Busy* and *Active* are set to FALSE only after *Execute* is set to FALSE.

Initial state

The outputs *Done, InGear, InSync, InVelocity, Error, ErrorID* and *CommandAborted* are reset with a falling edge at input *Execute*, if the function block is not active (Busy=FALSE). However, a falling edge at *Execute* has no influence on the command execution. Resetting *Execute* during command execution ensures that one of the outputs is set at the end of the command for a PLC cycle. Only then are the outputs reset.

If Execute is triggered more than once while a command is executed, the function block will not execute further commands, without providing any feedback.

Input parameters

The input parameters are read with rising edge at Execute. To change the parameters the command has to be triggered again once it is completed, or a second instance of the function block must be triggered with new parameters during command execution.

If an input parameter is not transferred to the function block, the last value transferred to this block remains valid. A meaningful default value is used for the first call.

Position and Distance

The *Position* input designates a defined value within a coordinate system. *Distance,* in contrast, is a relative measurement, i.e. the distance between two positions. *Position* and *Distance* are specified in technical units, e.g. [mm] or [°], according to the axis scaling.

Dynamic parameters

The dynamic parameters for Move functions are specified in technical units with second as timebase. If an axis is scaled in millimeters, for example, the following units are used: *Velocity* [mm/s], *Acceleration* [mm/s^2], *deceleration* [mm/s^2], *jerk* [mm/s^3].

Error handling

All function blocks have two error outputs for indicating errors during command execution. *Error* indicates the error, *ErrorID* contains a supplementary error number. The outputs *Done, InVelocity, InGear* and *InSync* indicate successful command execution and are not set if *Error* becomes TRUE.

Errors of different type are signaled at the function block output. The error type is not specified explicitly. It depends on the unique, system-wide error number.

Error types

- Function block errors only concern the function block, not the axis (e.g. incorrect parameterization). Function block errors do not have to be reset explicitly. They are reset automatically when the *Execute* input is reset.

- Communication errors (the function block cannot address the axis, for example). Communication errors usually indicate incorrect configuration or parameterization. A reset is not possible. The function block can only be triggered again after the configuration was corrected.
- Axis errors (logical NC axis) usually occur during the motion (e.g. following error). They cause the axis to switch to error status. An axis error must be reset through MC_Reset [▶ 20].
- Drive errors (control device) may result in an axis error, i.e. an error in the logical NC axis. In many cases can axis errors and drive errors can be reset together through MC_Reset. [▶ 20] Depending on the drive controller, a separate reset mechanism may be required (e.g. connection of a reset line to the control device).

Behavior of the *Done* output

The *Done* output (or alternatively *InVelocity, InGear, InSync* etc.) is set when a command was executed successfully. If several function blocks are used for an axis and the running command is interrupted through a further block, the *Done* output for the first block is not set.

Behavior of the *CommandAborted* output

CommandAborted is set if a command is interrupted through another block.

Behavior of the *Busy* output

The *Busy* output indicates that the function block is active. The block can only be triggered with a rising edge at *Execute*, if *Busy* is FALSE. *Busy* is immediately set with a rising edge at *Execute* and is only reset when the command was completed successful or unsuccessfully. As long as *Busy* is TRUE, the function block must be called cyclically for the command to be executed.

Behavior of the *Active* output

If the axis movement is controlled by several functions, the Active output of each block indicates that the axis executes the command. The status Busy=TRUE and Active=FALSE means that the command is not or no longer executed.

Enable input and *Valid* output

In contrast to *Execute* the *Enable* input results in an action being executed permanently and repeatedly, as long as *Enable* is TRUE. MC_ReadStatus [▶ 30]cyclically updates the status of an axis, for example, as long as *Enable* is TRUE. A function block with an *Enable* input indicates through the *Valid* output that the data indicated at the outputs are valid. The data can be updated continuously while *Valid* is TRUE.

BufferMode

Some function blocks have a *BufferMode* input for controlling the command flow with several function blocks. For example, *BufferMode* can specify that a command interrupts another command (non-queued mode) or that the following command is only executed after the previous command (queued mode). In queued mode *BufferMode* can be used to specify the movement transition from one command to the next. This is referred to as *Blending*, which specifies the velocity at the transition point.

A second function block is required to use the buffer mode. It is not possible to trigger a move block with new parameters while it is active.

In non-queued mode a subsequent command leads to termination of a running command. In this case the previous command sets the *CommandAborted* output. In queued mode a subsequent command waits until a running command is completed. Note here that an endless movement (MC_MoveVelocity) does not permit a queued subsequent command. Queued commands always lead immediately to an endless movement being aborted, as in non-queued operation.

Only one command is queued while another command is executed. If more than one command is triggered during a running command, then the last-started command to be queued is rejected with an error (error 0x4292 Buffer Full). If the last command is started in non-queued mode (*Aborting*), it becomes active and interrupts the running and an already queued command.

BufferModes

- Aborting : Default mode without buffering. The command is executed immediately and interrupts any other command that may be running.
- Buffered : The command is executed once no other command is running on the axis. The previous movement continues until it has stopped. The following command is started from standstill.
- BlendingLow: The command is executed once no other command is running on the axis. In contrast to *Buffered* the axis does not stop at the previous target, but passes through this position with the lower velocity of two commands.
- BlendingHigh The command is executed once no other command is running on the axis. In contrast to *Buffered* the axis does not stop at the previous target, but passes through this position with the higher velocity of two commands.
- BlendingNext : The command is executed once no other command is running on the axis. In contrast to *Buffered* the axis does not stop at the previous target, but passes through this position with the velocity of the last command.
- BlendingPrevious: The command is executed once no other command is running on the axis. In contrast to *Buffered* the axis does not stop at the previous target, but passes through this position with the velocity of the first command.

Diagram of the buffer modes [▶ 103]

Optional blending position

Blending in the different buffer modes takes place in each case at the target position of the currently running command. In the case of *MoveVelocity* no target position is defined and in other cases it may be useful to change the blending position. To do this a *BlendingPosition* can be defined via the *Options* input of the function block (see below), which is then used for the new command. The optional *BlendingPosition* must be located before the target position of the previous command, otherwise the new command will be rejected with an error message (0x4296). If the optional *BlendingPosition* has already been driven past, then the new command is instantly implemented and thus behaves like an *Aborting* command.

Options input

Many function blocks have an *Options* input with a data structure containing additional, infrequently required options. For the basic block function these options are often not required, so that the input can remain open. The user only has to populate the *Options* data structure in cases where the documentation explicitly refers to certain options.

Slave axes

Travel commands can be applied to coupled slave axes, if this option was explicitly activated in the axis parameters. Travel commands can be applied to coupled slave axes, if this option was explicitly activated in the axis parameters. In this case only *Buffer-ModeAborting* is *possible*.

5 Migration from TcMC to TcMC2

The main differences and modifications between the TcMC motion control library and the extended TcMC2 library are listed here, so that the effort for converting an existing project can be estimated.

Axis data structure

In the past an axis required two data structures for cyclic data exchange with the NC.

NcToPlc_Axis1 AT %I* : NCTOPLC_AXLESTRUCT;

PlcToNc_Axis1 AT %Q* : PLCTONC_AXLESTRUCT;

In most function blocks, including MC_MoveAbsolute [▶ 58], the NCTOPLC_AXLESTRUCT data structure was transferresd at the *Axis* input. Certain function blocks, including MC_Power [▶ 19], expected an additional PLCTONC_AXLESTRUCT structure.

In the TcMC2 environment the axis structure was extended so that all required data are included in a single structure, which is transferred to each MC function block.

Axis1: AXIS_REF [▶ 101];

The structure contains the cyclic input and output data for the NC plus additional status information. An existing project generally accesses the content of the NcToPlc structure. The data are also available in the *Axis1* structure and can be used to adapt the application program.

Example:

TcMC : NcToPlc_Axis1.fPosSoll

TcMC2 : Axis1.NcToPlc.SetPos

Please note that the subelements for the NcToPlc and. PlcToNc structures now have English names in view of the international market. For example, the current set position for an axis is no longer referred to as *fPosSoll*, but as *SetPos*.

Function blocks

The input and output configuration of the function blocks has changed slightly compared with TcMC. The main new feature is support for MC_BufferMode [▶ 103] in Move blocks. In addition, the blocks now also support a Busy and Active output. These modifications generally only require little migration effort. The following table contains a list of function blocks with more extensive modifications.

TcMC	TcMC2	Note
MC_GearInFloat	MC_GearIn [▶ 93]	MC_GearIn now accepts the gear ratio as a floating point value
MC_NewPos MC_NewPosAndVelo	MC_Move...	The new *BufferMode* enables each Move block to be used to assign a new target for the axis or change the velocity. The NewPos function blocks are therefore no longer required.
MC_MoveAbsoluteOrRestart	MC_Move...	MoveAbsoluteOrRestart can be replaced with two instances of a Move block (see *BufferMode*).
MC_CamIn MC_CamInExt	MC_CamIn	The new MC_CamIn function block deals with the function of the extended MC_CamInExt block. The input circuit was adapted accordingly.

MC_SetReferenceFlag	MC_Home [▶ 88]	Setting and resetting of the reference flag (axis is referenced) can be achieved with the MC_Home block.
MC_SetPositionOnTheFly	MC_SetPosition [▶ 21]	For actual value setting on the fly, MC_SetPosition is used in relative mode (mode=TRUE).
MC_SetActualPosition	MC_SetPosition [▶ 21]	MC_SetActualPosition is replaced with MC_SetPosition. The new function block sets the actual and set positions.
MC_GearOutExt	MC_Move...	Travel commands can be applied to coupled slave axes, if this option was explicitly activated in the axis parameters (from TwinCAT 2.11). Travel commands can be applied to coupled slave axes, if this option was explicitly activated in the axis parameters. In this case only *Buffer-ModeAborting* is *possible*.
MC_OrientedStop	MC_MoveModulo [▶ 64]	MC_MoveModulo can be started from standstill or during motion. In the latter case the block behaves like *MC_OrientedStop*
MC_Stop	MC_Halt [▶ 78] , MC_Stop [▶ 80]	MC_Halt executes a normal stop during motion. In contrast, *MC_Stop* locks the axis and prevents further travel commands. It should only be used in special situations.
MC_Home	MC_Home [▶ 88]	MC_Home transfers the *bCalibrationCam* signal of the homing sensor only while the block is active. To execute homing from the System Manager with F9, the signal must be transferred to the NC by other means, e.g. through direct allocation: *Axis.PlcToNc.ControlDword.5 := HomingSensor;*

TcNC library

The previous TcMC library required declarations and functions from the TcNC library, so that this was always integrated in a project. The new TcMC2 library no longer has this dependency. All required declarations and functions are now included in TcMC2 library itself, so that the TcNC library is no longer required. Nevertheless, the TcNC library can still be used for compatibility reasons.

Status information

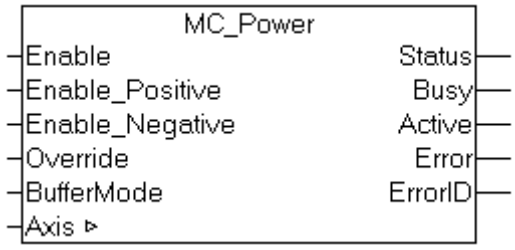
In existing motion applications axis status information was often determined via a function call (*AxisHasJob(), AxisIsMoving()* etc.). While these functions can still be used if the TcNC library is integrated, we now recommended a different approach:

The complete status information for an axis is included in the above-mentioned axis data structure *Axis1*:AXIS_REF [▶ 101]. However, these data have to be updated cyclically by calling the function block *MC_ReadStatus* or an *Axis1.ReadStatus* action at the start of the PLC cycle. Current status information is then available at any point in the program during the PLC cycle.

6 Organisation function blocks

6.1 Axis functions

6.1.1 MC_Power



MC_Power activates software enable for an axis. Enable can be activated for both directions of travel or only one direction. At *Status* output operational readiness of the axis is indicated.

A velocity override influences the velocity of all travel commands by a specified percentage.

In addition to software enable it may be necessary to activate a hardware enable signal in order to enable a drive. This signal is not influenced by MC_Power and must be activated separately by the PLC.

Depending on the drive type, Status also signals operational readiness of the drive. Digital drives provide feedback on operational readiness, while analog drives are unable to indicate their operational readiness. In the latter case Status only indicated operational readiness of the control side.

Inputs

```
VAR_INPUT
Enable          : BOOL; (* B *)
Enable_Positive : BOOL; (* E *)
Enable_Negative : BOOL; (* E *)
Override        : LREAL (* V *) := 100.0; (* in percent - Beckhoff proprietary input *)
BufferMode      : MC_BufferMode; (* V *)
END_VAR
```

MC_BufferMode [▶ 103]

Enable	General software enable for the axis.
Enable_Positive	Feed enable in positive direction. Only takes effect if *Enable* = TRUE.
Enable_Negative	Feed enable in negative direction. Only takes effect if *Enable* = TRUE.
Override	Velocity override in % for all movement commands. (0 ≤*Override*≤ 100.0)
BufferMode	The BufferMode is evaluated if *Enable* is reset. *MC_Aborting* mode leads to immediate deactivation of the axis enable. Otherwise, e.g. in *MC_Buffered* mode, the function block waits until the axis no longer executes a command.

General rules for MC function blocks [▶ 14]

Outputs

```
VAR_OUTPUT
Status    : BOOL; (* B *)
Busy      : BOOL; (* V *)
Active    : BOOL; (* V *)
Error     : BOOL; (* B *)
ErrorID   : UDINT; (* E *)
END_VAR
```

Status	Status=TRUE indicates that the axis is ready for operation.
Busy	The *Busy* output is TRUE, as long as the function block is called up with *Enable* = TRUE
Active	Active indicates that the command is executed.
Error	Becomes TRUE if an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number

General rules for MC function blocks [▶ 14]

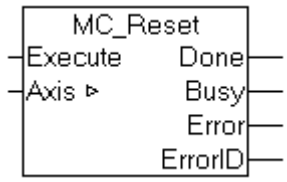
Inputs/outputs

```
VAR_IN_OUT
Axis : AXIS_REF;
END_VAR
```

Axis	Axis data structure

The axis data structure of type AXIS_REF [▶ 101] addresses an axis unambiguously within the system. Among other parameters it contains the current axis status, including position, velocity or error state.

6.1.2 MC_Reset



An axis reset is carried out with the function block *MC_Reset*.

MC_Reset initially resets the NC axis. In many cases this also leads to a reset of the connected drive units. Depending on the bus system or drive types, in some cases a separate reset may be required for the drive units.

Inputs

```
VAR_INPUT
Execute : BOOL;
END_VAR
```

Execute	The command is executed with a rising edge at input *Execute*.

Outputs

```
VAR_OUTPUT
Done    : BOOL;
```

```
Busy    : BOOL;
Error   : BOOL;
ErrorID : UDINT;
END_VAR
```

Done	The *Done* output becomes TRUE when the reset was carried out successfully.
Busy	The *Busy* output becomes TRUE when the command is started with *Execute* and remains TRUE as long as the command is processed. If *Busy* becomes FALSE again, the function block is ready for a new command. At the same time one of the outputs, *Done* or *Error*, is set.
Error	Becomes TRUE if an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number

Inputs/outputs

```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Axis data structure

The axis data structure of type AXIS_REF [▶ 101] addresses an axis uniquely within the system. Among other parameters it contains the current axis status, including position, velocity or error status.

6.1.3 MC_SetPosition

MC_SetPosition sets the current axis position to a parameterizable value.

In absolute mode, the actual position is set to the parameterized absolute *Position value*. In relative mode, the actual position is offset by the parameterized *Position* value. In both cases, the set position of the axis is set such that any following error that may exist is retained. The switch *Options.ClearPositionLag* can be used to clear the following error.

Relative mode can be used to change the axis position during the motion.

Inputs

```
VAR_INPUT
Execute  : BOOL;
Position : LREAL;
Mode     : BOOL; (* RELATIVE=True, ABSOLUTE=False (Default)*)
Options  : ST_SetPositionOptions;
END_VAR
```

Execute	The command is executed with a rising edge at input *Execute*.
Position	Position value to which the axis position is to be set. In absolute mode the actual position is set to this value, in relative mode it is shifted by this value.
Mode	The axis position is set to an absolute value set if *Mode*=FALSE. Otherwise is the axis position is changed relative to the specified *Position* value. Relative mode can be used for changing the position of an axis during motion.
Options	The data structure option includes additional, rarely required parameters. The input can normally remain open.

Options.	**ClearPositionLag**	ClearPositionLag can optionally be used to set the set and actual positions to the same value. In this case the following error is cancelled.
Options.	**SelectEncoderIndex**	SelectEncoderIndex can optionally be set if an axis with several encoders is used and the position of a certain encoder is to be set (*Options.EncoderIndex*).
Options.	**EncoderIndex**	EncoderIndex indicates the encoder (0 to n) if *SelectEncoderIndex* is TRUE.

General rules for MC function blocks [▶ 14]

Outputs

```
VAR_OUTPUT
Done    : BOOL;
Busy    : BOOL;
Error   : BOOL;
ErrorID : UDINT;
END_VAR
```

Done	The *Done* output becomes TRUE, once the position was set successfully.
Busy	The *Busy* output becomes TRUE when the command is started with *Execute* and remains TRUE as long as the command is processed. If *Busy* becomes FALSE again, the function block is ready for a new job. At the same time one of the outputs, *Done* or *Error*, is set.
Error	Becomes TRUE if an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number

General rules for MC function blocks [▶ 14]

Inputs/outputs

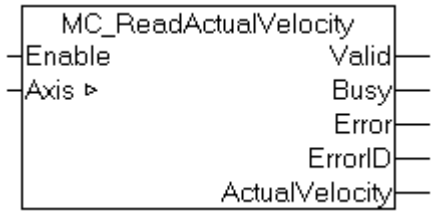
```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Axis data structure

The axis data structure of type AXIS_REF [▶ 101] addresses an axis uniquely within the system. Among other parameters it contains the current axis status, including position, velocity or error status.

6.2 Status and parameter

6.2.1 MC_ReadActualVelocity



The actual axis position can be read with the function block *MC_ReadActualVelocity*.

Inputs

```
VAR_INPUT
Enable   : BOOL;
END_VAR
```

Enable	The command is executed as long as *Enable* is active.

Outputs

```
VAR_OUTPUT
Valid          : BOOL;
Busy           : BOOL;
Error          : BOOL;
ErrorID        : UDINT;
ActualVelocity : LREAL;
END_VAR
```

Valid	Indicates that *ActualVelocity* is valid.
Busy	Indicates that the function block is active.
Error	Becomes TRUE if an error occurs.
ErrorID	If the *error* output is set, this parameter supplies the error number
ActualVelocity	Current axis velocity

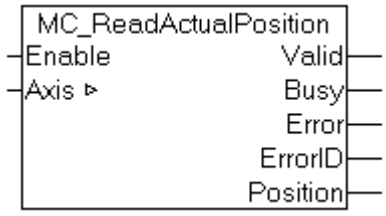
Inputs/outputs

```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Axis data structure

The axis data structure of type AXIS_REF [▶ 101] addresses an axis uniquely within the system. Among other parameters it contains the current axis status, including position, velocity or error status.

6.2.2 MC_ReadActualPosition



The current axis position can be read with the function block *MC_ActualPosition*.

Inputs

```
VAR_INPUT
Enable   : BOOL;
END_VAR
```

Enable	The command is executed as long as *Enable* is active.

Outputs

```
VAR_OUTPUT
Valid    : BOOL;
Busy     : BOOL;
Error    : BOOL;
ErrorID  : UDINT;
Position : LREAL;
END_VAR
```

Valid	Indicates that the *Position* output is valid.
Busy	Indicates that the function block is active.
Error	Becomes TRUE if an error occurs.
ErrorID	If the *error* output is set, this parameter supplies the error number
Position	Current axis position

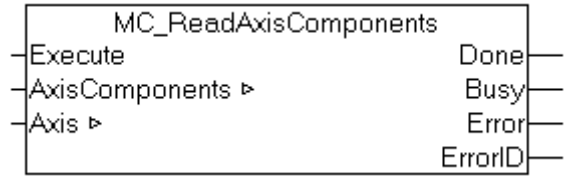
Inputs/outputs

```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Axis data structure

The axis data structure of type AXIS_REF [▶ 101] addresses an axis uniquely within the system. Among other parameters it contains the current axis status, including position, velocity or error status.

6.2.3 MC_ReadAxisComponents



The function block *MC_ReadAxisComponents* is used to read information relating to the subelements encoder, drive and controller of an axis.

In this case "axis" refers to the TwinCAT NC axis and not the drive.

Inputs

```
VAR_INPUT
Execute  : BOOL;
END_VAR
```

Execute	The command is executed with the rising edge.

Outputs

```
VAR_OUTPUT
Done    : BOOL;
Busy    : BOOL;
Error   : BOOL;
ErrorID : UDINT;
END_VAR
```

Done	Becomes TRUE, if the parameters were read successfully.
Busy	The *Busy* output becomes TRUE when the command is started with *Execute* and remains TRUE as long as the command is processed. If *Busy* becomes FALSE again, the function block is ready for a new order. At the same time one of the outputs, *Done* or *Error*, is set.
Error	Becomes TRUE if an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number

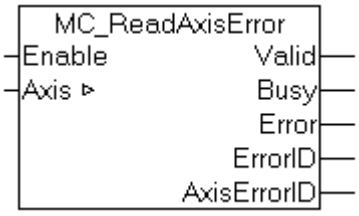
Inputs/outputs

```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Axis data structure

The axis data structure of type AXIS_REF [▶ 101] addresses an axis unambiguously within the system. Among other parameters it contains the current axis status, including position, velocity or error state.

6.2.4 MC_ReadAxisError



MC_ReadAxisError reads the axis error of an axis.

Inputs

```
VAR_INPUT
Enable  : BOOL; (* B *)
END_VAR
```

Enable	The axis error is output at the *AxisErrorID* output as long as *Enable* is active

General rules for MC function blocks [▶ 14]

Outputs

```
VAR_OUTPUT
Valid       : BOOL; (* B *)
Busy        : BOOL; (* E *)
Error       : BOOL; (* B *)
ErrorID     : DWORD; (* B *)
AxisErrorID : DWORD; (* B *)
END_VAR
```

Valid	The error signaled at the AxisErrorID output is valid
Busy	The *Busy* output becomes TRUE when the command is started with *Enable* and remains TRUE as long as the command is processed. If *Busy* becomes FALSE again, the function block is ready for a new job.
Error	Becomes TRUE if an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number
AxisErrorID	Error number for the axis

General rules for MC function blocks [▶ 14]

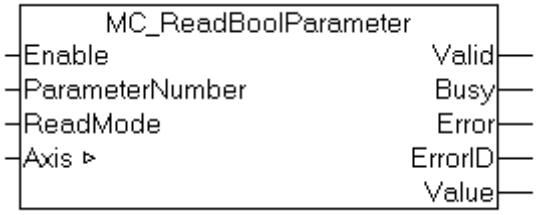
Inputs/outputs

```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Axis data structure

The axis data structure of type AXIS_REF [▶ 101] addresses an axis uniquely within the system. Among other parameters it contains the current axis status, including position, velocity or error status.

6.2.5 MC_ReadBoolParameter



The function block *MC_ReadBoolParameter* is used to read a boolean axis parameter.

In this case "axis" refers to the TwinCAT NC axis and not the drive.

Inputs

```
VAR_INPUT
Enable          : BOOL; (* B *)
ParameterNumber : MC_AxisParameter; (* B *)
ReadMode        : E_ReadMode (* V *)
END_VAR
```

E_ReadMode [▶ 108] MC_AxisParameter [▶ 110]

Enable	The command is executed as long as *Enable* is active.
ParameterNumber	Number [▶ 110] of the parameter to be read.
ReadMode	Read mode [▶ 108] of the parameter to be read (once or cyclic).

Outputs

```
VAR_OUTPUT
Valid   : BOOL; (* B *)
Busy    : BOOL; (* E *)
Error   : BOOL; (* B *)
ErrorID : DWORD;(* E *)
Value   : BOOL; (* B *)
END_VAR
```

Valid	The value signaled at the *Value* output is valid
Busy	The *Busy* output becomes TRUE when the command is started with *Enable* and remains TRUE as long as the command is processed. If *Busy* becomes FALSE again, the function block is ready for a new order.
Error	Becomes TRUE if an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number
Value	Displays the boolean value that was read.

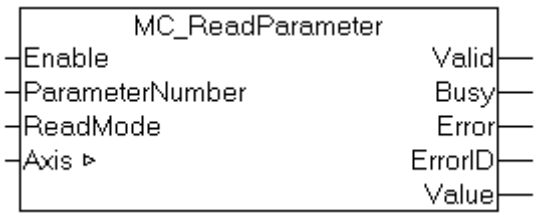
Inputs/outputs

```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Axis data structure

The axis data structure of type AXIS_REF [▶ 101] addresses an axis unambiguously within the system. Among other parameters it contains the current axis status, including position, velocity or error state.

6.2.6 MC_ReadParameter



The function block *MC_ReadParameter* is used to read an axis parameter.

In this case "axis" refers to the TwinCAT NC axis and not the drive.

Inputs

```
VAR_INPUT
Enable          : BOOL; (* B *)
ParameterNumber : MC_AxisParameter; (* B *)
ReadMode        : E_ReadMode (* V *)
END_VAR
```

E_ReadMode [▶ 108] MC_AxisParameter [▶ 110]

Enable	The command is executed as long as *Enable* is active.
ParameterNumber	Number [▶ 110] of the parameter to be read.
ReadMode	Read mode [▶ 108] of the parameter to be read (once or cyclic).

Outputs

```
VAR_OUTPUT
Valid   : BOOL;  (* B *)
Busy    : BOOL;  (* E *)
Error   : BOOL;  (* B *)
ErrorID : DWORD; (* E *)
Value   : LREAL; (* B *)
END_VAR
```

Valid	The value signaled at the Value output is valid
Busy	The *Busy* output becomes TRUE when the command is started with *Enable* and remains TRUE as long as the command is processed. If *Busy* becomes FALSE again, the function block is ready for a new order.
Error	Becomes TRUE if an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number
Value	Displays the read value.

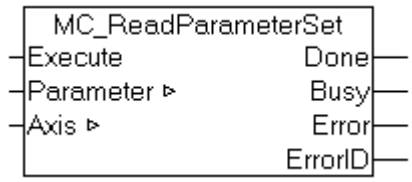
Inputs/outputs

```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Axis data structure

The axis data structure of type AXIS_REF [▶ 101] addresses an axis unambiguously within the system. Among other parameters it contains the current axis status, including position, velocity or error state.

6.2.7 MC_ReadParameterSet



The complete parameter set of an axis can be read with the function block *MC_ReadParameterSet*.

In this case "axis" refers to the TwinCAT NC axis and not the drive.

Inputs

```
VAR_INPUT
Execute : BOOL;
END_VAR
```

Execute	The command is executed with the rising edge.

Outputs

```
VAR_OUTPUT
Done    : BOOL;
Busy    : BOOL;
Error   : BOOL;
ErrorID : UDINT;
END_VAR
```

Done	Becomes TRUE, if the parameters were read successfully.
Busy	The *Busy* output becomes TRUE when the command is started with *Execute* and remains TRUE as long as the command is processed. If *Busy* becomes FALSE again, the function block is ready for a new order. At the same time one of the outputs, *Done* or *Error*, is set.
Error	Becomes TRUE if an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number

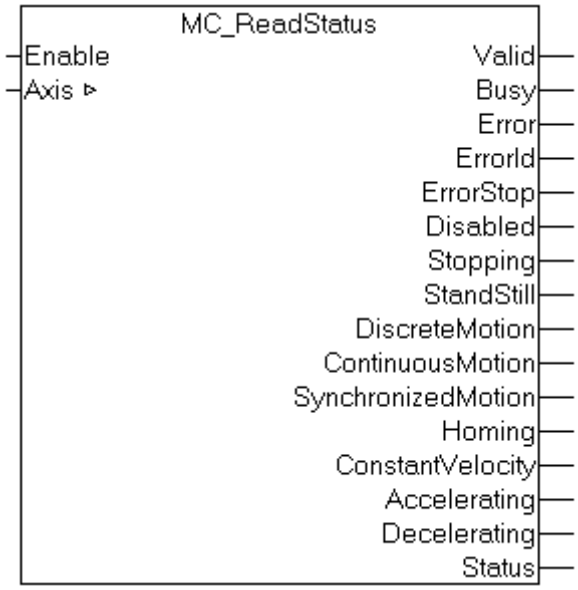
Inputs/outputs

```
VAR_IN_OUT
Parameter : ST_AxisParameterSet;
Axis      : AXIS_REF;
END_VAR
```

Parameter	Parameter data structure [▶ 111] into which the parameters are read
Axis	Axis data structure

The axis data structure of type AXIS_REF [▶ 101] addresses an axis unambiguously within the system. Among other parameters it contains the current axis status, including position, velocity or error state.

6.2.8 MC_ReadStatus



MC_ReadStatus determines the current operating state of an axis and signals it at the function block outputs.

The updated operating state is additionally stored in the *Status* output data structure and in the *Axis.Status* axis data structure. This means the operating state only has to be read once at the start of each PLC cycle and can then be accessed via *Axis.Status*.

The *Axis* variable (type AXIS_REF) already includes an instance of the function block *MC_ReadStatus*. This means that the operating state of an axis can be updated at the start of a PLC cycle by calling up Axis.ReadStatus.

Sample:

```
PROGRAM MAIN
VAR
Axis1   : AXIS_REF
END_VAR

(* call the read status function *)
Axis1.ReadStatus;
```

Inputs

```
VAR_INPUT
Enable  : BOOL;
END_VAR
```

Enable	As long as *Enable* = TRUE, the axis operating state is updated with each call of the function block.

General rules for MC function blocks [▶ 14]

Outputs

```
VAR_OUTPUT
Valid               : BOOL;
Busy                : BOOL;
Error               : BOOL;
ErrorId             : UDINT;
(* motion control statemachine states: *)
ErrorStop           : BOOL;
```

```
Disabled            : BOOL;
Stopping            : BOOL;
StandStill          : BOOL;
DiscreteMotion      : BOOL;
ContinuousMotion    : BOOL;
SynchronizedMotion  : BOOL;
Homing              : BOOL;
(* additional status *)
ConstantVelocity    : BOOL;
Accelerating        : BOOL;
Decelerating        : BOOL;
(* status data structure *)
Status              : ST_AxisStatus;
END_VAR
```

Valid	Indicates that the axis operating state indicated at the other outputs is valid.
Busy	Indicates that the function block is active.
Error	Becomes TRUE if an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number
ErrorStop	Axis status according to the PlcOpen state diagram [▶ 11]
Disabled	Axis status according to the PlcOpen state diagram [▶ 11]
Stopping	Axis status according to the PlcOpen state diagram [▶ 11]
StandStill	Axis status according to the PlcOpen state diagram [▶ 11]
DiscreteMotion	Axis status according to the PlcOpen state diagram [▶ 11]
ContinousMotion	Axis status according to the PlcOpen state diagram [▶ 11]
SynchronizedMotion	Axis status according to the PlcOpen state diagram [▶ 11]
Homing	Axis status according to the PlcOpen state diagram [▶ 11]
ConstantVelocity	The axis is moving with constant velocity
Acceleration	The axis accelerates.
Decelerating	The axis decelerates.
Status	Extended status data structure [▶ 109] with additional status information.

General rules for MC function blocks [▶ 14]

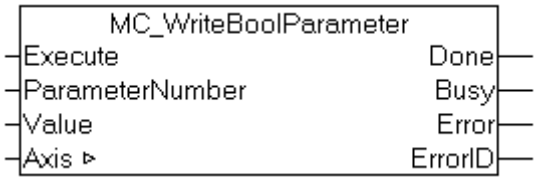
Inputs/outputs

```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Axis data structure

The axis data structure of type AXIS_REF [▶ 101] addresses an axis unambiguously within the system. Among other parameters it contains the current axis status, including position, velocity or error state.

6.2.9 MC_WriteBoolParameter



Boolean parameters for the axis can be written with the function block *MC_WriteBoolParameter*.

In this case "axis" refers to the TwinCAT NC axis and not the drive.

Inputs

```
VAR_INPUT
Execute         : BOOL;
ParameterNumber : INT;
Value           : BOOL;
END_VAR
```

Execute	The command is executed with the rising edge.
ParameterNumber	Number [▶ 110] of the parameter to be written.
Value	This BOOL value is written.

Outputs

```
VAR_OUTPUT
Done    : BOOL;
Busy    : BOOL;
Error   : BOOL;
ErrorID : UDINT;
END_VAR
```

Done	Becomes TRUE, if the parameters were written successfully.
Busy	The *Busy* output becomes TRUE when the command is started with *Execute* and remains TRUE as long as the command is processed. If *Busy* becomes FALSE again, the function block is ready for a new order. At the same time one of the outputs, *Done* or *Error*, is set.
Error	Becomes TRUE if an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number

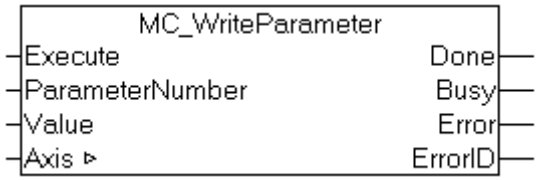
Inputs/outputs

```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Axis data structure

The axis data structure of type AXIS_REF [▶ 101] addresses an axis unambiguously within the system. Among other parameters it contains the current axis status, including position, velocity or error state.

6.2.10 MC_WriteParameter



Parameters for the axis can be written with the function block *MC_WriteParameter*.

In this case "axis" refers to the TwinCAT NC axis and not the drive.

Inputs

```
VAR_INPUT
Execute         : BOOL;
ParameterNumber : INT;
Value           : LREAL;
END_VAR
```

Execute	The command is executed with the rising edge.
ParameterNumber	Number [▶ 110] of the parameter to be written.
Value	This LREAL value is written.

Outputs

```
VAR_OUTPUT
Done    : BOOL;
Busy    : BOOL;
Error   : BOOL;
ErrorID : UDINT;
END_VAR
```

Done	Becomes TRUE, if the parameters were written successfully.
Busy	The *Busy* output becomes TRUE when the command is started with *Execute* and remains TRUE as long as the command is processed. If *Busy* becomes FALSE again, the function block is ready for a new order. At the same time one of the outputs, *Done*, *CommandAborted* or *Error*, is set.
Error	Becomes TRUE if an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number

Inputs/outputs

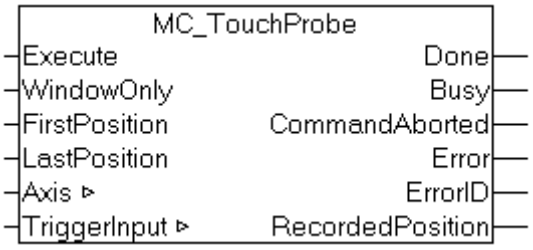
```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Axis data structure

The axis data structure of type AXIS_REF [▶ 101] addresses an axis unambiguously within the system. Among other parameters it contains the current axis status, including position, velocity or error state.

6.3 Touch probe

6.3.1 MC_TouchProbe



The *MC_TouchProbe* function block records an axis position at the point in time of a digital signal (measuring probe function). The position is usually not recorded directly in the PLC environment, but via an external hardware latch, and is thus very accurate and independent of cycle time. The function block controls this mechanism and determines the externally recorded position.

Prerequisites

The prerequisite for the position acquisition is suitable encoder hardware that is able to latch the recorded position. The following equipment is supported, for example: SERCOS drives, the Beckhoff AX2000 with SERCOS and Lightbus interfaces and the Beckhoff KL5101 Encoder Bus Terminals. The digital trigger signal is wired into this hardware and, independently of the PLC cycle, triggers the recording of the current axis position.

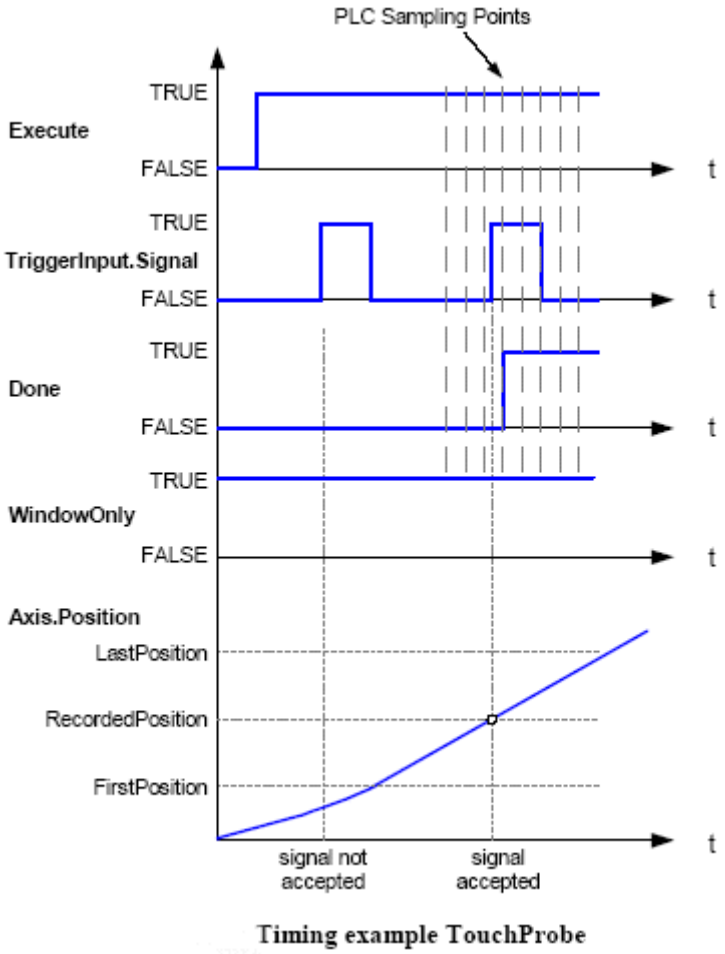
These end devices have to be configured to some extent so that a position recording is possible. For details, read Measuring probe evaluation with AX2xxx-B200 (Lightbus), Measuring probe evaluation with AX2xxx-B750 (SERCOS), AX5000 Probe Unit and AX5000 Function of a probe unit.

Restrictions

Irrespective of the hardware used, MC_TouchProbe can only record one edge of a probe unit at a time. If both edges are to be recorded, for example, then the block must be restarted after the first edge with a changed parameterization. Edges that follow one another in quick succession therefore cannot be recorded. In order to avoid this problem, you are referred to the extended function block MC_TouchProbe_V2 [▶ 37].

After a measuring probe cycle has been initiated by a rising edge on the *Execute* input, this is only terminated if the outputs *Done*, *Error* or *CommandAborted* become TRUE. If the process is to be interrupted at an intermediate point in time, the function block MC_AbortTrigger [▶ 40] with the same TriggerInput [▶ 114] data structure must be called up. Otherwise no new cycle can be initiated.

Signal curve



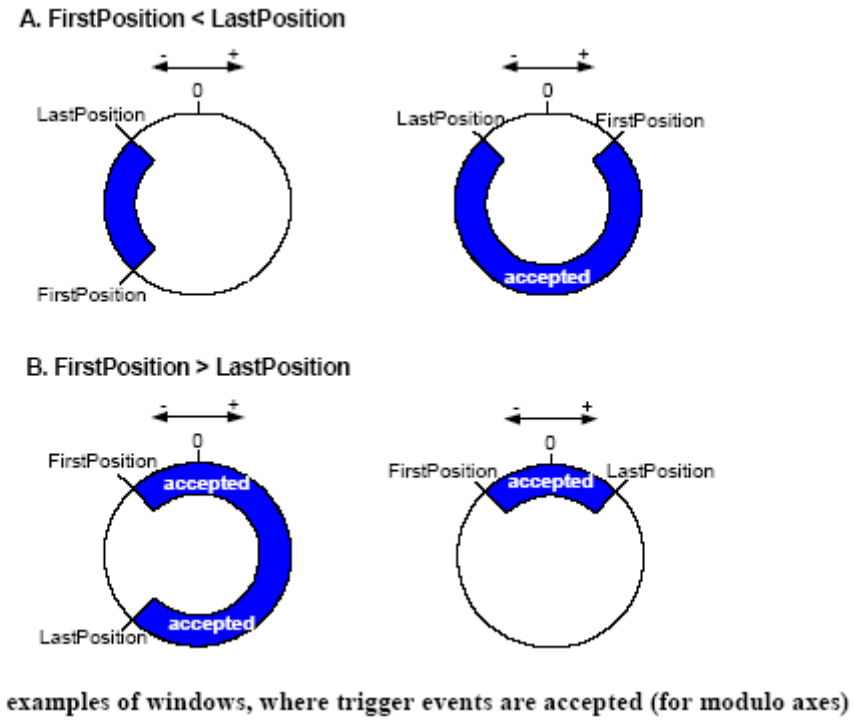
Timing example TouchProbe

Inputs

```
VAR_INPUT
Execute       : BOOL;
WindowOnly    : BOOL;
FirstPosition : LREAL;
LastPosition  : LREAL;
END_VAR
```

Execute	The command is executed with the rising edge and the external position latch is activated.
WindowOnly	If this option is active, only one position inside the window between *FirstPosition* and LastPosition is recorded. Positions outside the window are rejected and the external position latch is automatically newly activated. Only if the recorded position lies inside the window does *Done* become TRUE. The recording window can be interpreted in terms of absolute or modulo values. In this connection the flag ModuloPositions [▶ 114] in the structure TriggerInput [▶ 114] is to be set accordingly. In the case of absolute value positions there is exactly one window. In the case of modulo value positions the window repeats itself within the modulo cycle defined in the axis parameters (e.g. 0 to 360 degrees).
FirstPosition	Initial position of the recording window, if WindowOnly is TRUE. This position can be interpreted as an absolute or modulo value. In this

	connection the flag ModuloPositions [▶ 114] is to be set appropriately in the structure *TriggerInput* (see below).
LastPosition	Final position of the recording window, if WindowOnly is TRUE. This position can be interpreted as an absolute or modulo value. In this connection the flag ModuloPositions [▶ 114] is to be set appropriately in the structure *TriggerInput* (see below).



examples of windows, where trigger events are accepted (for modulo axes)

Outputs

```
VAR_OUTPUT
Done              : BOOL;
Busy              : BOOL;
CommandAborted    : BOOL;
Error             : BOOL;
ErrorID           : UDINT;
RecordedPosition  : LREAL;
END_VAR
```

Done	Becomes TRUE, if an axis position has been recorded successfully. The position is sent to the output *RecordedPosition*.
Busy	Becomes TRUE as soon as the function block is active, and becomes FALSE when it has returned to its initial state.
CommandAborted	Becomes TRUE if the process is interrupted by an external event, e.g. by the call up of MC_AbortTrigger [▶ 40].
Error	Becomes TRUE, as soon as an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number
RecordedPosition	Axis position recorded at the point in time of the trigger signal

Inputs/outputs

```
VAR_IN_OUT
Axis         : AXIS_REF;
TriggerInput : TRIGGER_REF;
END_VAR
```

Axis	Axis data structure [▶ 101]
TriggerInput	TRIGGER. REF [▶ 114] data structure for describing the trigger source

6.3.2 MC_TouchProbe_V2

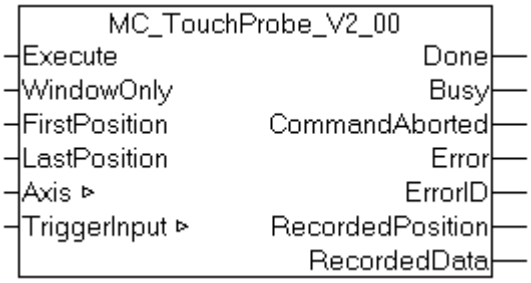


Fig. 1: MC_TouchProbe_V2_00

The *MC_TouchProbe_V2* function block records an axis position at the point in time of a digital signal (measuring probe function). The position is usually not recorded directly in the PLC environment, but via an external hardware latch, and is thus very accurate and independent of cycle time. The function block controls this mechanism and determines the externally recorded position.

The function of the *MC_TouchProbe_V2* function block is similar that of the *MC_TouchProbe* function block. With several instances, however, it is possible to operate up to two probe units at the same time and in parallel to record the rising and falling signal edges each with an instance. Furthermore, a continuous mode is available that evaluates successive signal edges without renewed activation.

Prerequisites

- TwinCAT version 2.11 R2 build 2022 or higher (before that use MC_TouchProbe [▶ 34])

The prerequisite for the position acquisition is suitable encoder hardware that is able to latch the recorded position. Support is offered for:

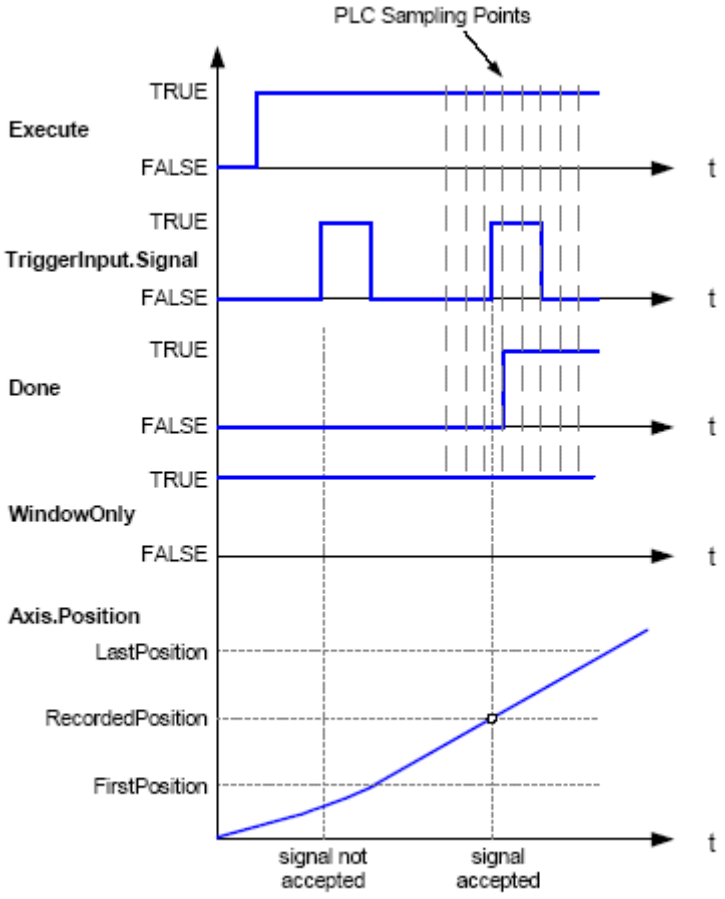
- SERCOS drives
  In contrast to MC_TouchProbe, the drive must be configured with an extended interface, in which the parameters S 0 0405 and S-0 0406 are included in the process image. See also ...
- EtherCAT SoE drives (E.g. AX5000)
  In contrast to MC_TouchProbe, the drive must be configured with an extended interface, in which the parameters S 0 0405 and S-0 0406 are included in the process image. See also ...
- EtherCAT CoE drives
  The drive must be configured with the parameter 0x60B9 (touch probe status) in the process image.
- EL5101, KL5101
  Latching of the C track and the digital input is possible. This hardware can only record one signal or edge at a time. Continuous mode is not supported.

The digital trigger signal is wired into this hardware and, independently of the PLC cycle, triggers the recording of the current axis position.

These end devices have to be configured to some extent so that a position recording is possible. Beckhoff EtherCAT drives can be configured with the System Manager. Note that the probe unit has to be configured with the "Extended NC Probe Unit" interface.

After a measuring probe cycle has been initiated by a rising edge on the *Execute* input, this is only terminated if the outputs *Done*, *Error* or *CommandAborted* become TRUE. If the process is to be interrupted at an intermediate point in time, the function block MC_AbortTrigge_V2 [▶ 41] with the same TriggerInput [▶ 114] data structure must be called up. Otherwise no new cycle can be initiated.

Signal curve



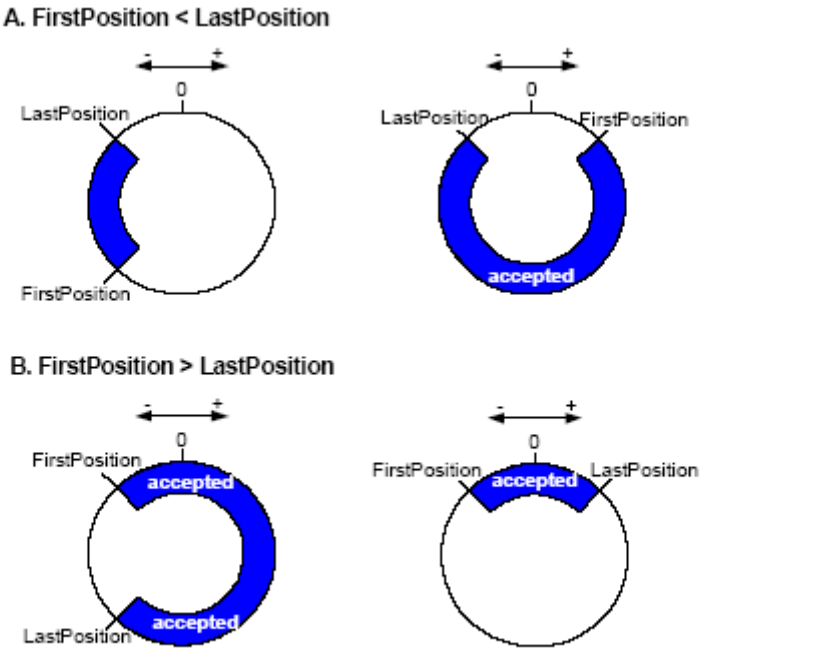
Timing example TouchProbe

Inputs

```
VAR_INPUT
Execute : BOOL;
WindowOnly : BOOL;
FirstPosition : LREAL;
LastPosition  : LREAL;
END_VAR
```

Execute	The command is executed with the rising edge and the external position latch is activated.
WindowOnly	If this option is active, only one position inside the window between *FirstPosition* and LastPosition is recorded. Positions outside the window are rejected and the external position latch is automatically newly activated. Only if the recorded position lies inside the window does *Done* become TRUE. The recording window can be interpreted in terms of absolute or modulo values. In this connection the flag ModuloPositions [▶ 114] in the structure TriggerInput [▶ 114] is to be set accordingly. In the case of absolute value positions there is exactly one window.

	In the case of modulo value positions the window repeats itself within the modulo cycle defined in the axis parameters (e.g. 0 to 360 degrees).
FirstPosition	Initial position of the recording window, if WindowOnly is TRUE. This position can be interpreted as an absolute or modulo value. In this connection the flag ModuloPositions [▶ 114] is to be set appropriately in the structure *TriggerInput* (see below).
LastPosition	Final position of the recording window, if WindowOnly is TRUE. This position can be interpreted as an absolute or modulo value. In this connection the flag ModuloPositions [▶ 114] is to be set appropriately in the structure *TriggerInput* (see below).



examples of windows, where trigger events are accepted (for modulo axes)

Outputs

```
VAR_OUTPUT
Done              : BOOL;
Busy              : BOOL;
CommandAborted    : BOOL;
Error             : BOOL;
ErrorID           : UDINT;
RecordedPosition : LREAL;
RecordedData : MC_TouchProbeRecordedData;
END_VAR
```

Done	Becomes TRUE, if an axis position has been recorded successfully. The position is sent to the output *RecordedPosition*.
Busy	Becomes TRUE as soon as the function block is active, and becomes FALSE when it has returned to its initial state.
CommandAborted	Becomes TRUE if the process is interrupted by an external event, e.g. by the call up of MC_AbortTrigger [▶ 40].
Error	Becomes TRUE, as soon as an error occurs.

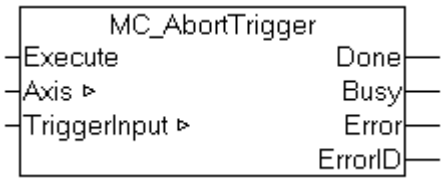
ErrorID	If the error output is set, this parameter supplies the error number.
RecordedPosition	Axis position recorded at the point in time of the trigger signal
RecordedData	Data structure with complementary information relating to the logged axis position at the time of the trigger signal

Inputs/outputs

```
VAR_IN_OUT
Axis : AXIS_REF;
TriggerInput : TRIGGER_REF;
END_VAR
```

Axis	Axis data structure [▶ 101]
TriggerInput	TRIGGER_REF [▶ 114] data structure for describing the trigger source

6.3.3 MC_AbortTrigger



The *MC_AbortTrigger* function block interrupts a measuring probe cycle initiated by *MC_TouchProbe*. *MC_TouchProbe* initiates a measuring probe cycle by activating a position latch in external encoder or drive hardware. If the process is to be terminated before the trigger signal has activated the position latch, *MC_AbortTrigger* can be used for this purpose. If the measuring probe cycle has completed successfully, it is not necessary to call up this function block.

Inputs

```
VAR_INPUT
Execute  : BOOL;
END_VAR
```

Execute	The command is executed with the rising edge and the external position latch is deactivated.

Outputs

```
VAR_OUTPUT
Done    : BOOL;
Busy    : BOOL;
Error   : BOOL;
ErrorID : UDINT;
END_VAR
```

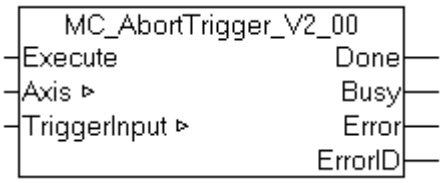
Done	Becomes TRUE, as soon as the measuring probe cycle has been interrupted successfully.
Busy	Becomes TRUE as soon as the function block is active, and becomes FALSE when it has returned to its initial state.
Error	Becomes TRUE, as soon as an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number.

Inputs/outputs

```
VAR_IN_OUT
Axis         : AXIS_REF;
TriggerInput : TRIGGER_REF;
END_VAR
```

Axis	Axis data structure [▶ 101]
TriggerInput	TRIGGER_REF [▶ 114] data structure for describing the trigger source. This data structure must be parameterized before the function block is called for the first time.

6.3.4 MC_AbortTrigger_V2



The *MC_AbortTrigger_V2* function block interrupts a measuring probe cycle initiated by *MC_TouchProbe_V2*. *MC_TouchProbe_V2* initiates a measuring probe cycle by activating a position latch in external encoder or drive hardware. If the process is to be terminated before the trigger signal has activated the position latch, *MC_AbortTrigger_V2* can be used for this purpose. If the measuring probe cycle has completed successfully, it is not necessary to call up this function block.

Inputs

```
VAR_INPUT
Execute  : BOOL;
END_VAR
```

Execute	The command is executed with the rising edge and the external position latch is deactivated.

Outputs

```
VAR_OUTPUT
Done     : BOOL;
Busy     : BOOL;
Error    : BOOL;
ErrorID  : UDINT;
END_VAR
```

Done	Becomes TRUE, as soon as the measuring probe cycle has been interrupted successfully.
Busy	Becomes TRUE as soon as the function block is active, and becomes FALSE when it has returned to its initial state.
Error	Becomes TRUE, as soon as an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number.

Inputs/outputs

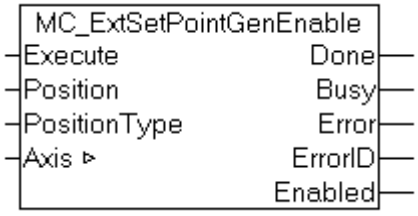
```
VAR_IN_OUT
Axis         : AXIS_REF;
TriggerInput : TRIGGER_REF;
END_VAR
```

Axis	Axis data structure [▶ 101]

TriggerInput	TRIGGER_REF [▶ 114] data structure for describing the trigger source. This data structure must be parameterized before the function block is called for the first time.

6.4 External set value generator

6.4.1 MC_ExtSetPointGenEnable



The external set value generator of an axis can be switched on with the function block *MC_ExtSetPointGenEnable*. The axis then adopts the set value specifications from its cyclic axis interface [▶ 102] (*Axis.PlcToNc.ExtSetPos, ExtSetVelo, ExtSetAcc* and *ExtSetDirection*).

An *external set value generator* is usually a PLC block that calculates cyclic set values for an axis and can therefore substitute the *internal set value generator* in an NC axis.

See also: MC_ExtSetPointGenDisable [▶ 43] and MC_ExtSetPointGenFeed [▶ 44]

Inputs

```
VAR_INPUT
Execute      : BOOL;
Position     : LREAL;
PositionType : E_PositionType;
END_VAR
```

Execute	The command is executed with the rising edge.
Position	Position for target position monitoring. Setting of this position does not mean that the axis moves to this position, for which only the external set value generator is responsible. Setting of this position activates target position monitoring, and the flag Data type ST_AxisStatus [▶ 109] becomes TRUE, as soon as this position is reached.
PositionType	Position type [▶ 115] - *POSITION TYPE_ABSOLUTE* or *POSITION TYPE_RELATIVE*

Outputs

```
VAR_OUTPUT
Done    : BOOL;
Busy    : BOOL;
Error   : BOOL;
ErrorID : UDINT;
Enabled : BOOL;
END_VAR
```

Done	Becomes TRUE, if the command was issued successfully.
Busy	Becomes TRUE as soon as the function block is active, and becomes FALSE when it has returned to its initial state.

Error	Becomes TRUE, as soon as an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number
Enabled	Enabled shows the current state of the external set value generator, independent of the function execution.

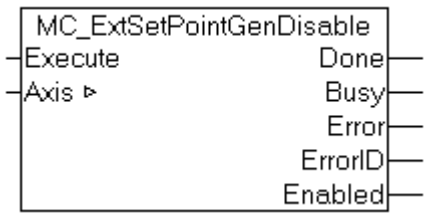
Inputs/outputs

```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Axis data structure [▶ 101]

The axis data structure of type AXIS_REF [▶ 101] addresses an axis uniquely within the system. Among other parameters it contains the current axis status, including position, velocity or error status.

6.4.2 MC_ExtSetPointGenDisable



The external set value generator of an axis can be switched off with the function block *MC_ExtSetPointGenDisable*. The axis then no longer adopts the set value specifications from its cyclic axis interface [▶ 102] (*Axis.PlcToNc.ExtSetPos, ExtSetVelo, ExtSetAcc* and *ExtSetDirection*)

An *external set value generator* is usually a PLC block that calculates cyclic set values for an axis and can therefore substitute the *internal set value generator* in an NC axis.

See also: MC_ExtSetPointGenEnable [▶ 42] and MC_ExtSetPointGenFeed [▶ 44]

Inputs

```
VAR_INPUT
Execute : BOOL;
END_VAR
```

Execute	The command is executed with the rising edge.

Outputs

```
VAR_OUTPUT
Done    : BOOL;
Busy    : BOOL;
Error   : BOOL;
ErrorID : UDINT;
Enabled : BOOL;
END_VAR
```

Done	Becomes TRUE, if the command was executed successfully.

Busy	Becomes TRUE as soon as the function block is active, and becomes FALSE when it has returned to its initial state.
Error	Becomes TRUE, as soon as an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number
Enabled	Enabled shows the current state of the external set value generator, independent of the function execution.

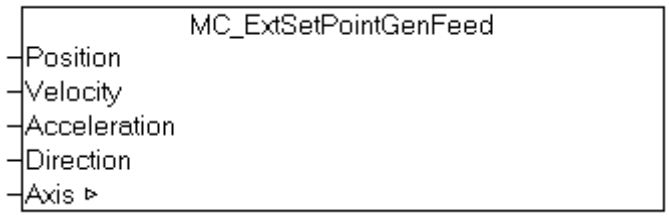
Inputs/outputs

```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Axis data structure

The axis data structure of type AXIS_REF [▶ 101] addresses an axis uniquely within the system. Among other parameters it contains the current axis status, including position, velocity or error status.

6.4.3 MC_ExtSetPointGenFeed



The *MC_ExtSetPointGenFeed* function is used to feed set values from an external set value generator into an axis. The function copies the data instantaneously into the cyclic axis interface [▶ 102] (*fExtSetPos, fExtSetVelo, fExtSetAcc* and *nExtSetDirection*). The function result of *MC_ExtSetPointGenFeed* is not used and therefore always FALSE.

An *external set value generator* is usually a PLC block that calculates cyclic set values for an axis and can therefore substitute the *internal set value generator* in an NC axis.

See also: MC_ExtSetPointGenEnable [▶ 42] and MC_ExtSetPointGenDisable [▶ 43]

Inputs

```
VAR_INPUT
Position     : LREAL;
Velocity     : LREAL;
Acceleration : LREAL;
Direction    : DINT;
END_VAR
```

Position	Set position from an external set value generator
Velocity	Set velocity from an external set value generator
Acceleration	Set acceleration from an external set value generator
Direction	Set direction from an external set value generator. (-1 = negative direction, 0 = standstill, 1 = positive direction)

Inputs/outputs

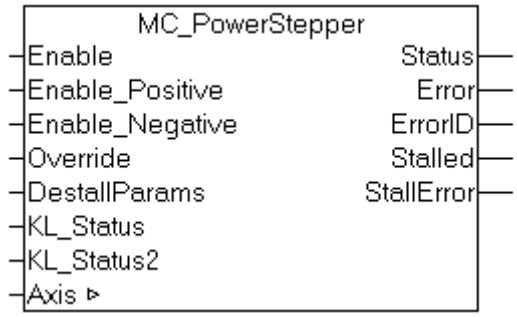
```
VAR_IN_OUT
Axis : AXIS_REF;
END_VAR
```

Axis	Axis data structure

The axis data structure of type AXIS_REF [▶ 101] addresses an axis uniquely within the system. Among other parameters it contains the current axis status, including position, velocity or error status.

6.5 Special extensions

6.5.1 MC_PowerStepper



The enables for an axis are set with the function block *MC_PowerStepper*. An *MC_Power* block is used internally for this purpose. The *MC_PowerStepper* also detects the stall situations that occur in stepper motors if they are overloaded, and offers suitable counter measures. The status bits of a KL2531 or KL2541 terminal are monitored, and the errors indicated there are reported to the NC.

There is more detailed explanation in the Appendix [▶ 46].

Inputs

```
VAR_INPUT
Enable          : BOOL;
Enable_Positive : BOOL;
Enable_Negative : BOOL;
Override        : LREAL;
DestallParams   : ST_PowerStepperStruct;
KL_Status       : USINT;
KL_Status2      : UINT;
END_VAR
```

Enable	NC controller enable for the axis.
Enable_Positive	NC advance movement enable in positive direction.
Enable_Negative	NC advance movement enable in negative direction.
Override	Override value in percent (e.g. 68.123%)
DestallParams	The functions of the block are enabled here [▶ 111], and their working rules are specified. ST_PowerStepperStruct [▶ 111]
KL_Status	The status byte of a terminal of type KL2531 or KL2541.
KL_Status2	The status word of a terminal of type KL2531 or KL2541.

Outputs

```
VAR_OUTPUT
Status     : BOOL;
Error      : BOOL;
ErrorID    : UDINT;
Stalled    : BOOL;
StallError : BOOL;
END_VAR
```

Status	Becomes TRUE once all enables were set successfully.
Error	Becomes TRUE if an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number.
Stalled	no description
StallError	no description

Inputs/outputs

```
VAR_IN_OUT
Axis : AXIS_REF;
END_VAR
```

Axis	Axis data structure

The axis data structure of type AXIS_REF [▶ 101] addresses an axis uniquely within the system. Among other parameters it contains the current axis status, including position, velocity or error status.

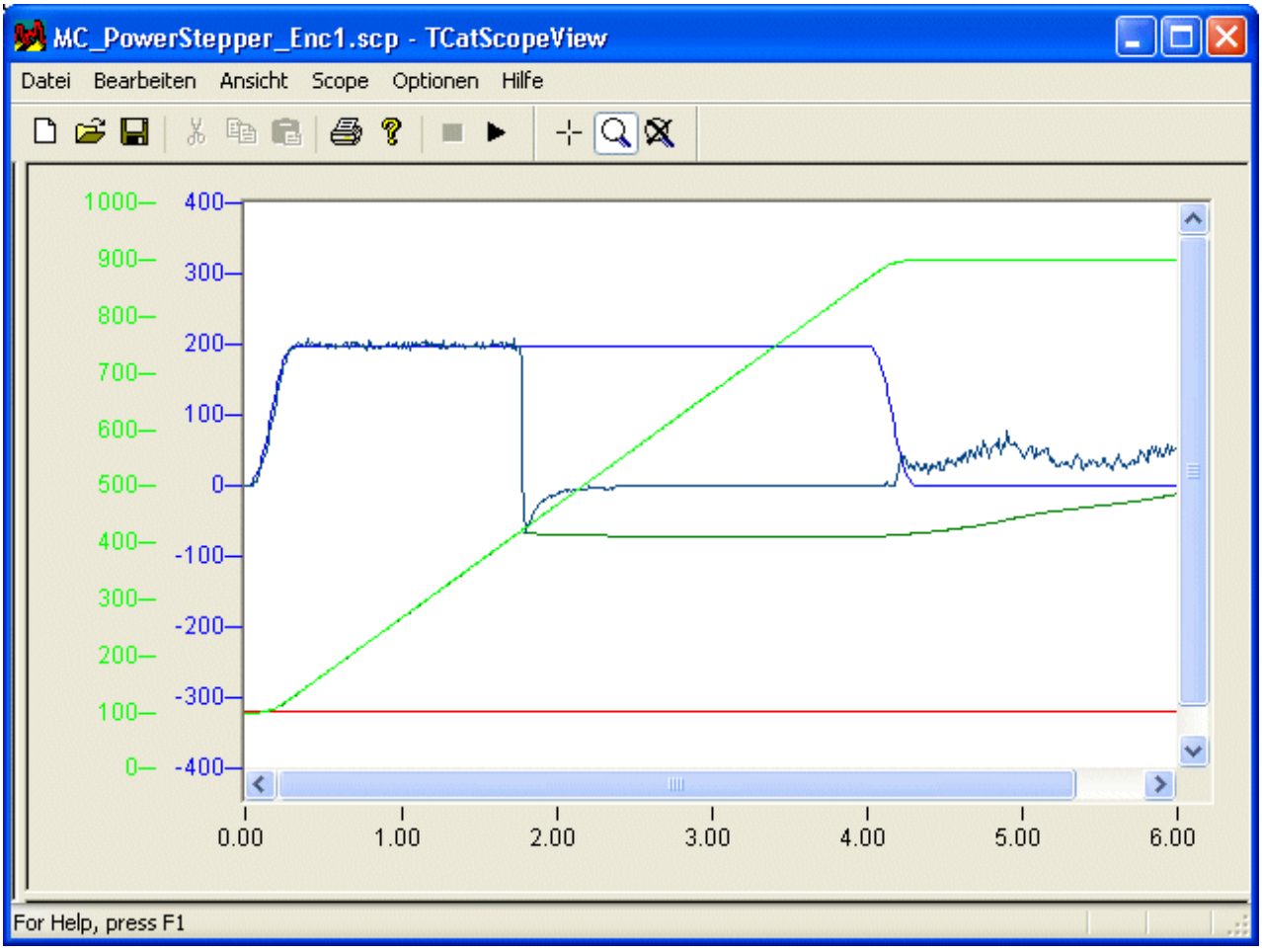
6.5.2 Notes on the MC_PowerStepper

The enables and the override for an axis are set with the MC_PowerStepper [▶ 45] function block. An MC_Power [▶ 19] block is used internally for this purpose. The *MC_PowerStepper* also detects the stall situations that occur in stepper motors if they are overloaded, and offers suitable counter measures. The status bits of a KL2531 or KL2541 terminal are monitored, and the errors indicated there are reported to the NC.

Stepper motor and synchronous servo: similarities and differences

Both types of motor use an electromagnetic field and the field of a permanent magnet in order to generate a driving force through their interaction. Whereas, however, the servomotor makes use of an expensive system of sensors in order to make specific adjustments to the alignments of the fields (current supplied dependent on the rotor position), this position-dependent control is not used for the stepper motor. This makes it possible to save considerable costs. There is, however, a possibility that some external force will push the motor beyond the position where it is able to generate the maximum torque. Because the electrically generated magnetic field does not take this into account, the restoring torque generated will fall as the excursion increases. As a result of this, if the excursion is more than the one half of one pole step then the corrective torque will change sign, pushing the motor on in the direction of the next pole position. Depending on the conditions that now apply, the motor may now latch into the new position (which means that a complete step has been lost), or the whole process may be repeated again here. The latter case is referred to as stalling, and is most likely to occur when current is fed to the motor at the typical frequency of the active drive operation.

Example 1: A stepper motor fitted with an encoder is operated with the NC PTP using the parameters typical for servos.

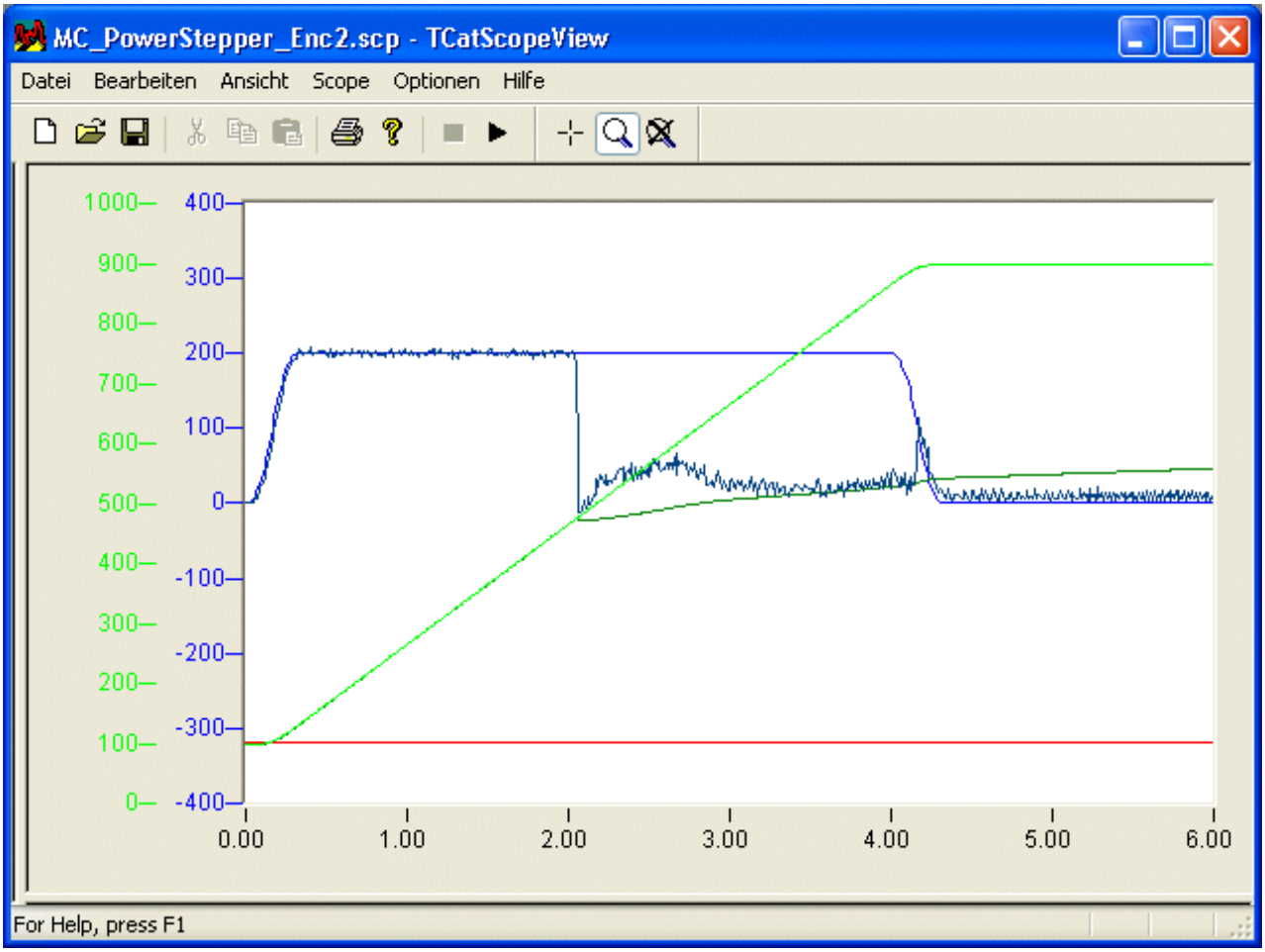


bStalled
SetVelo
ActualPos
ActualVelo
SetPos

After about 1.8 seconds, the axis is briefly blocked by an obstacle. Although the axis is then able to move freely, it is unable to follow the set value of the velocity, but will remain stationary, making considerable noise but without generating any detectable torque. Only after the profile generator has reached its target does the total of the set and correction velocities fall. In this example, the motor moves in an irregular manner. Even a small load torque will, however, prevent this. The only solution here would be to issue an MC_Reset [▶ 20] and to allow an appropriate settling time to pass. The axis would then have to be restarted by the application. A variety of state bits in the axis interface would react here. This must be appropriately considered in the application, as otherwise incorrect reactions may occur in the machine control process.

First corrective step: Controller limitation

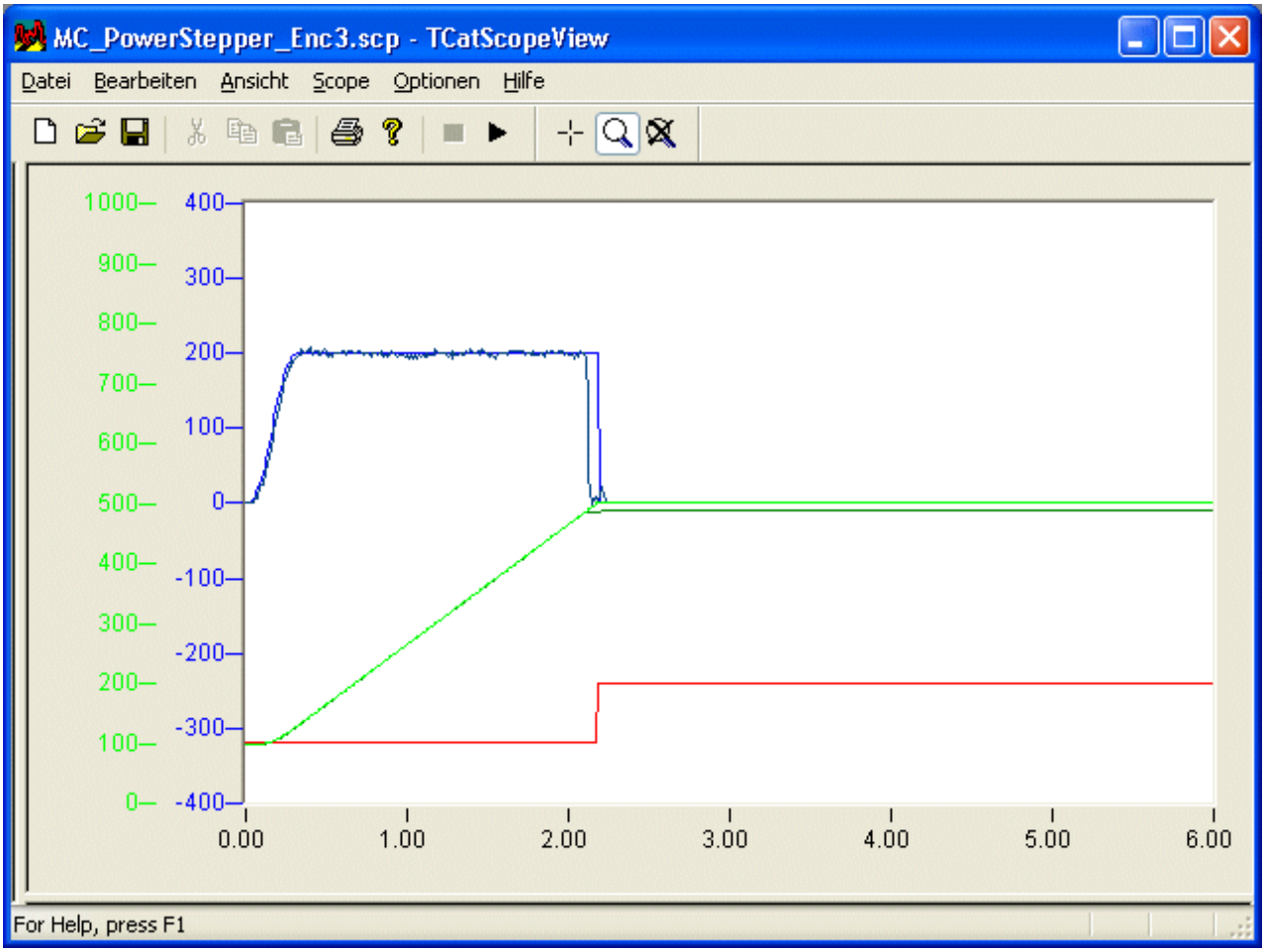
If, in the situation described above, the output of the position controller is limited to a sufficiently small value such as, for instance, 2%, the following pattern results.



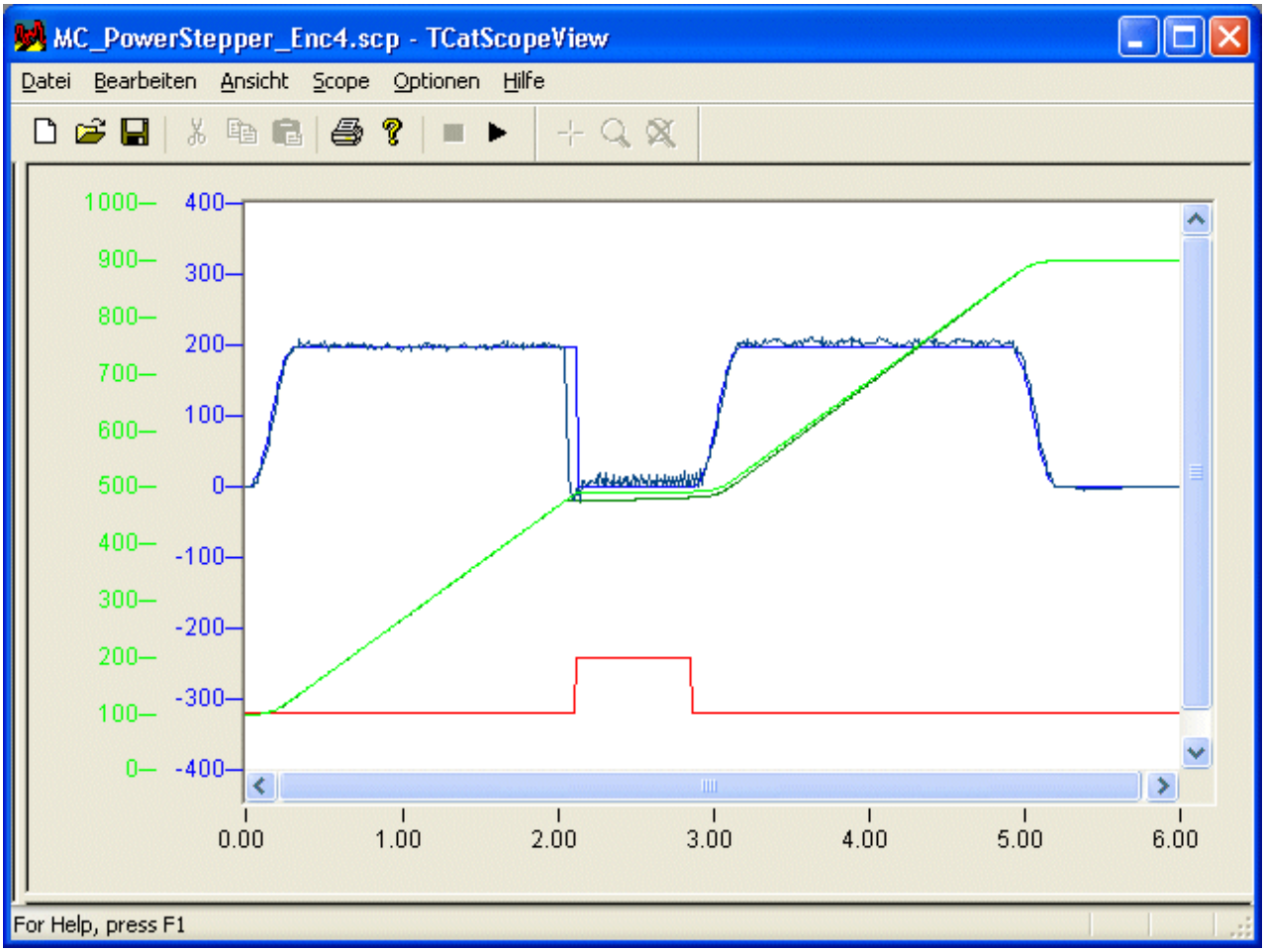
Here again, for the remaining period of profile generation, the set speed is too high for the stepper motor to be able to follow the set movement properly. When the end of the set profile has been reached, the stepper motor is now brought to its target by the position controller, at a working frequency that it is able to follow without the ramp. It generates a very high torque as it does this. The time required for this corrective measure is, however, very long.

Detection and handling of stall situations using an encoder

In order to be able to take appropriate counter-measures, it is first necessary to detect the problem. The following pattern results if an *MC_PowerStepper* function block is used. It has a parameter structure of type ST_PowerStepperStruct [▶ 111], in which PwStDetectMode_Lagging is entered as the DestallDetectMode. The block uses the following error of the axis as the basis of its decision, making use of the threshold value and the filter time from the NC axis data for the following error monitoring that is to be deactivated here. In this example, PwStMode_SetError is entered as the DestallMode. Initially, the only difference from the following error alarm is the different error code.



If PwStMode_UseOverride is entered as the DestallMode, the *MC_PowerStepper* block uses the override to halt the profile immediately. Because, however, this halt does not abort the profile, yet does at the same time prevent the end of the profile from being reached, there is no effect on any status bits. The controller output, limited here to 2%, brings the axis to within the following error threshold of the current set position of the profile. Then the override is then returned to the value specified by the application.



As a result, a significantly greater proportion of the overall profile is travelled at the specified speed, and the target position is reached correctly. The status information for the axis is generated correctly.

Combinations of stall detection and handling

The following table illustrates the combinations of the supported modes for stall detection and handling.

	PwStMode_SetError	PwStMode_SetErrNonRef	PwStMode_UseOverride
PwStDetectMode_Encoderless	Comment 1	Comment 2	not suitable
PwStDetectMode_Lagging	Comment 3	not useful	Comment 4

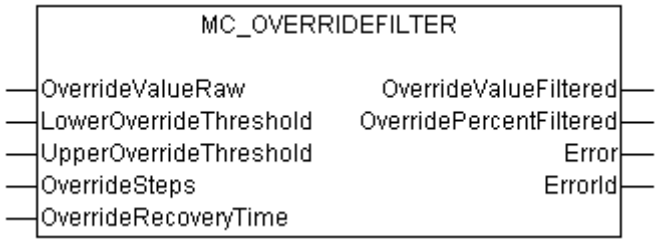
Comment 1: Useful for axes without encoder that are not referenced.

Comment 2: Useful for axes without encoder that are referenced with the aid of the terminal's pulse counter and, for instance, an external sensor.

Comment 3: The resultant behavior largely corresponds to following error monitoring.

Comment 4: Useful for axes with encoder.

6.5.3 MC_OverrideFilter



The function block *MC_OverrideFilter* can be used to convert an unfiltered override value consisting of digits (e.g. a voltage value of an analog input terminal) into a filtered override value that matches the cyclic axis interface (PlcToNc) (DWORD in the range 0...1000000). This filtered override is also available in percent (LREAL in the range 0...100%).
The raw input value is limited to a validity range by *LowerOverrideThreshold* and *UpperOverrideThreshold*, and implemented as parameterizable steps (resolution) (*OverrideSteps*). After each override change at the output of the FB, a minimum recovery time is awaited internally (*OverrideRecoveryTime*) before a new override value can be accepted. The only exceptions are the override values 0% and 100%, which are always implemented without delay for safety reasons.

Due to the gradation of the override output value (*OverrideValueFiltered*), the filtered override may become zero for very small override input values (*OverrideValueRaw*). A zero override leads to standstill of the axis. If total standstill is undesired, *OverrideValueRaw* should not fall below the smallest level.

Inputs

```
VAR_INPUT
OverrideValueRaw       : DINT;
LowerOverrideThreshold : DINT := 0;     (* 0...32767 digits *)
UpperOverrideThreshold : DINT := 32767;(* 0...32767 digits *)
OverrideSteps          : UDINT := 200; (* 200 steps=> 0.5 percent*)
OverrideRecoveryTime   : TIME := T#150ms; (* 150 ms *)
END_VAR
```

OverrideValueRaw	Raw, unfiltered override value (e.g. a voltage value of an analog input terminal).
LowerOverrideThreshold	The lower threshold for the raw override value.
UpperOverrideThreshold	The upper threshold for the raw override value.
OverrideSteps	The specified steps (override resolution).
OverrideRecoveryTime	Minimum recovery time, after which a new filtered override value is placed on the output. The override values 0% and 100% are implemented without delay.

Outputs

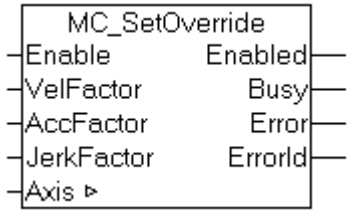
```
VAR_OUTPUT
OverrideValueFiltered   : DWORD; (* 0...1000000 counts *)
OverridePercentFiltered : LREAL; (* 0...100 % *)
Error                   : BOOL;
ErrorId                 : UDINT;
END_VAR
```

OverrideValueFiltered	The filtered override value in digits (the data type matches the override in the cyclic axis interface 0 to 1000000).
OverridePercentFiltered	The filtered override value in percent (0..100%).
Error	Becomes TRUE if an error occurs.

ErrorID	If the error output is set, this parameter supplies the error number.

Possible error number	Possible causes
MC_ERROR_PARAMETER_NOT_CORRECT	• OverrideSteps <= 1 • LowerOverrideThreshold >= UpperOverride-Threshold

6.5.4 MC_SetOverride



The override for an axis can be specified with the function block *MC_SetOverride*.

Inputs

```
VAR_INPUT
Enable     : BOOL; (* B *)
VelFactor  : LREAL (* B *) := 1.0; (*1.0 = 100%*)
AccFactor  : LREAL (* E *) := 1.0; (*1.0 = 100%*)(* not supported *)
JerkFactor : LREAL (* E *) := 1.0; (*1.0 = 100%*)(* not supported *)
END_VAR
```

Enable	The command is executed as long as *Enable* is active.
VelFactor	Velocity override factor
AccFactor	not supported
JerkFactor	not supported

Outputs

```
VAR_OUTPUT
Enabled : BOOL;
Busy    : BOOL;
Error   : BOOL;
ErrorID : UDINT;
END_VAR
```

Enabled	The parameterized override is set
Busy	The *Busy* output becomes TRUE when the command is started with *Enable* and remains TRUE as long as the command is processed. If *Busy* becomes FALSE again, the function block is ready for a new job.
Error	Becomes TRUE if an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number.

Inputs/outputs

```
VAR_IN_OUT
Axis   : AXIS_REF;
END_VAR
```

Axis	Axis data structure

The axis data structure of type AXIS_REF [▶ 101] addresses an axis uniquely within the system. Among other parameters it contains the current axis status, including position, velocity or error status.

6.5.5 MC_SetEncoderScalingFactor

MC_SetEncoderScalingFactor changes the scaling factor for the active encoder of an axis, either at standstill or in motion.

The change can be absolute or relative. This mode is only suitable at standstill, since in absolute mode the change in scaling factor leads to a position discontinuity. In relative mode an internal position offset is adapted at the same time such that no discontinuity occurs. Please note that intervention during motion changes the actual velocity of the axis while the real velocity remains constant. Therefore only small changes can be implemented during the motion.

Inputs

```
VAR_INPUT
Execute       : BOOL;
ScalingFactor : LREAL;
Mode          : E_SetScalingFactorMode;
Options : ST_SetEncoderScalingOptions;
END_VAR
```

Execute	The command is executed with a rising edge at input *Execute*.	
ScalingFactor	Scaling factor of the active encoder of an axis. The scaling factor is specified in physical positioning units [u] divided by the number of encoder increments.	
Mode	The scaling factor can be set in absolute or relative mode *(ENCODERSCALINGMODE_ABSOLUTE, ENCODERSCALINGMODE_RELATIVE).* In absolute mode counting starts at the origin of the axis coordinate system, resulting in a position discontinuity if the scaling factor is changed. In relative mode the actual position of the axis does not change. This *mode* is therefore also suitable for changes during motion.	
Options	The data structure option includes additional, rarely required parameters. The input can normally remain open.	
Options.	**SelectEncoderIndex**	SelectEncoderIndex can optionally be set if an axis with several encoders is used and the position of a certain encoder is to be set (*Options.EncoderIndex*).
Options.	**EncoderIndex**	EncoderIndex indicates the encoder (0 to n) if *SelectEncoderIndex* is TRUE.

General rules for MC function blocks [▶ 14]

Outputs

```
VAR_OUTPUT
Done    : BOOL;
Busy    : BOOL;
Error   : BOOL;
ErrorID : UDINT;
Options : ST_SetPositionOptions;
END_VAR
```

Done	The *Done* output becomes TRUE, once the position was set successfully.

Busy	The *Busy* output becomes TRUE when the command is started with *Execute* and remains TRUE as long as the command is processed. If *Busy* becomes FALSE again, the function block is ready for a new job. At the same time one of the outputs, *Done* or *Error*, is set.
Error	Becomes TRUE if an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number:

General rules for MC function blocks [▶ 14]

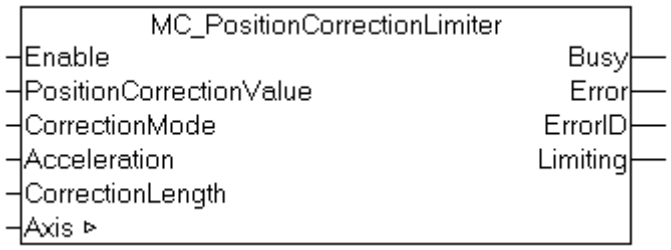
Inputs/outputs

```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Axis data structure

The axis data structure of type AXIS_REF [▶ 101] addresses an axis uniquely within the system. Among other parameters it contains the current axis status, including position, velocity or error status.

6.5.6 MC_PositionCorrectionLimiter



The function block *MC_PositionCorrectionLimiter* writes a correction value (*PositionCorrectionValue*) at the actual position of an axis. Depending on the correction mode the data are fed either directly or filtered to the axis.

VAR_INPUT

```
VAR_INPUT
Enable                  : BOOL;
PositionCorrectionValue : LREAL;
CorrectionMode          : E_AxisPositionCorrectionMode;
Acceleration            : LREAL;
CorrectionLenght        : LREAL;
END_VAR
```

Enable	The continuous writing of the *PositionCorrectionValue* is activated by this input. It must be TRUE as long as new correction values are to be accepted.
PositionCorrectionValue	The correction value that is to be added to the actual value of the axis.
CorrectionMode	Depending on this mode the *PositionCorrectionValue* is written either directly or filtered. For a detailed description see E_AxisPositionCorrectionMode [▶ 113].

Acceleration	Depending on the *CorrectionMode* the maximum acceleration to reach the new correction value is specified here. In the case of PositionCorrectionMode_Fast [▶ 113] this value has a direct effect on the position delta by PLC-tick. Max. permissible correction value position delta = acceleration * (PLC cycle time)^2. The position correction is not limited if acceleration is parameterized to 0.0.
CorrectionLength	If the *CorrectionMode* corresponds to PositionCorrectionMode_FullLength [▶ 113], this parameter becomes active. A change in the *PositonCorrectionValue* is distributed over this correction length.

VAR_IN_OUT

```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	AXIS_REF [▶ 101] axis data structure

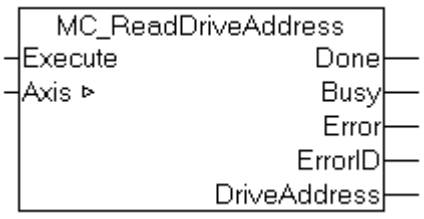
VAR_OUTPUT

```
VAR_OUTPUT
Busy      : LREAL;
Error     : BOOL;
ErrorId   : UDINT;
Limiting : BOOL;
ND_VAR
```

Busy	Goes TRUE as soon as the function block is active and FALSE when it returns to the original state.
Error	Becomes TRUE, as soon as an error occurs.
ErrorId	If the error output is set, this parameter supplies the error number.
Limiting	Goes TRUE if the demanded *PositionCorrectionValue* has not yet been completely accepted.

The *Position Correction* parameter in the System Manager must be enabled in order to use this function block successfully.

6.5.7 MC_ReadDriveAddress



MC_ReadDriveAddress reads the ADS access data for a drive device connected to the axis. This information is required for accessing the device, e.g. for special parameterization.

Inputs

```
VAR_INPUT
Execute : BOOL; (* B *)
END_VAR
```

Execute	The command is executed with a rising edge at input *Execute*.

General rules for MC function blocks [▶ 14]

Outputs

```
VAR_OUTPUT
Done          : BOOL;   (* B *)
Busy          : BOOL;   (* E *)
Error         : BOOL;   (* B *)
ErrorID       : DWORD;  (* B *)
DriveAddress : ST_DriveAddress;  (* B *)

END_VAR
```

Done	Becomes TRUE if the command was executed error-free.
Busy	The *Busy* output becomes TRUE when the command is started with *Execute* and remains TRUE as long as the command is processed. If *Busy* becomes FALSE again, the function block is ready for a new job.
Error	Becomes TRUE if an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number.
DriveAddress	ADS access data [▶ 111] of a drive unit connected to the axis.

General rules for MC function blocks [▶ 14]

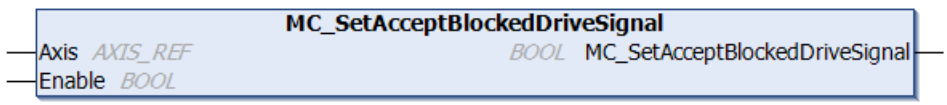
Inputs/outputs

```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Axis data structure

The axis data structure of type AXIS_REF [▶ 101] addresses an axis uniquely within the system. Among other parameters it contains the current axis status, including position, velocity or error status.

6.5.8 MC_SetAcceptBlockedDriveSignal



There are situations in which a drive no longer follows the NC setpoints, e.g. if an axis reaches a limit switch. The NC interprets such a situation as an error, and the drive is stopped. In some cases the user may want to provoke such a situation deliberately, e.g. in order to move to a limit switch for a reference run. The function MC_SetAcceptBlockedDriveSignal can be used to temporarily prevent the NC axis generating an error in situations where the drive no longer follows the NC setpoints.

- See also bit 8 of the ControlDWord in AXIS_REF.

- A SERCOS/SoE drive reports "Drive follows the command values" via status bit 3 of drive status word S-0-0135.
- A CANopen/CoE drive reports "Drive follows the command values" via status bit 12 of object 6041h.

FUNCTION MC_SetAcceptBlockedDriveSignal: BOOL

Inputs

```
VAR_INPUT
    Enable :  BOOL;
END_VAR
```

Enable: NC controller enable for the axis.

Inputs/outputs

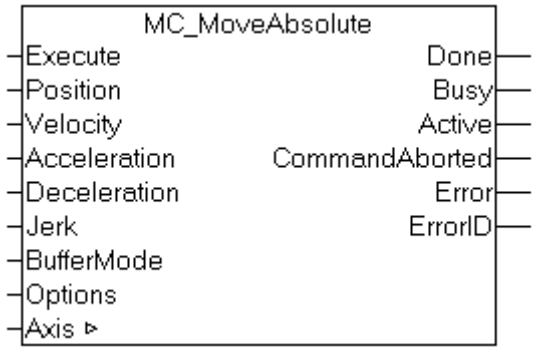
```
VAR_IN_OUT
    Axis : AXIS_REF;
END_VAR
```

Axis: Axis data structure that unambiguously addresses an axis in the system. Among other parameters it contains the current axis status, including position, velocity or error state. (Type: AXIS_REF [▶ 101])

7 Motion function blocks

7.1 Point to point motion

7.1.1 MC_MoveAbsolute



MC_MoveAbsolute starts positioning to an absolute target position and monitors the axis movement over the whole travel path. The Done output is set once the target position has been reached. Otherwise the CommandAborted or, on error, the Error output is set.

MC_MoveAbsolute is predominantly used for linear axis systems. For modulo axes the position is not interpreted as a modulo position, but as an absolute position in continuous absolute coordinate system. Alternatively, the MC_MoveModulo [▶ 64] block can be used for modulo positioning.

Travel commands can be applied to coupled slave axes, if this option was explicitly activated in the axis parameters. Travel commands can be applied to coupled slave axes, if this option was explicitly activated in the axis parameters. In this case only *Buffer-ModeAborting* is *possible*.

Inputs

```
VAR_INPUT
Execute      : BOOL;
Position     : LREAL;
Velocity     : LREAL;
Acceleration : LREAL;
Deceleration : LREAL;
Jerk         : LREAL;
BufferMode   : MC_BufferMode;
Options      : ST_MoveOptions;
END_VAR
```

Execute	The command is executed with a rising edge at input *Execute*.
Position	Absolute target position to be used for positioning.
Velocity	Maximum travel velocity (>0).
Acceleration	Acceleration (≥0). If the value is 0, the standard acceleration from the axis configuration in the System Manager is used.
Deceleration	Deceleration (≥0). If the value is 0, the standard deceleration from the axis configuration in the System Manager is used.
Jerk	Jerk (≥0). If the value is 0, the standard jerk from the axis configuration in the System Manager is used.
BufferMode	The BufferMode [▶ 103] is analyzed, if the axis is already executing another command. The running command can be aborted, or the new command

	becomes active after the running command. The BufferMode also determines the transition condition from the current to the next command. If the command is applied to a coupled slave axis used, the only available buffer mode is *Aborting*. A second function block is required to use the buffer mode. It is not possible to trigger a move block with new parameters while it is active.
Options	The data structure option includes additional, rarely required parameters. The input can normally remain open.

General rules for MC function blocks [▶ 14]

Outputs

```
VAR_OUTPUT
Done            : BOOL;
Busy            : BOOL;
Active          : BOOL;
CommandAborted : BOOL;
Error           : BOOL;
ErrorID         : UDINT;
END_VAR
```

Done	The *Done* output becomes TRUE once the target position was reached.
Busy	The *Busy* output becomes TRUE when the command is started with *Execute* and remains TRUE as long as the movement command is processed. If *Busy* becomes FALSE again, the function block is ready for a new job. At the same time one of the outputs, *Done*, *CommandAborted* or *Error*, is set.
Active	Active indicates that the command is executed If the command was queued, it becomes active once a running command is completed.
CommandAborted	Becomes TRUE, if the command could not be fully executed. The axis may have been stopped, or the running command may have been followed by a further Move command.
Error	Becomes TRUE if an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number.

General rules for MC function blocks [▶ 14]

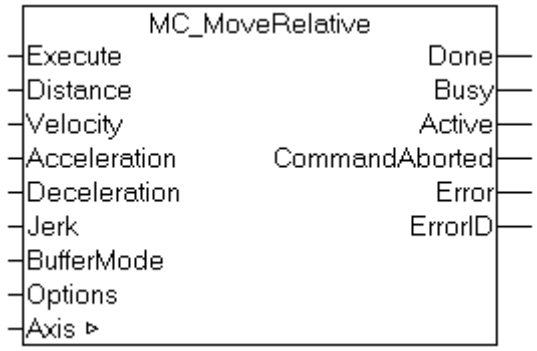
Inputs/outputs

```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Axis data structure

The axis data structure of type AXIS_REF [▶ 101] addresses an axis uniquely within the system. Among other parameters it contains the current axis status, including position, velocity or error status.

7.1.2 MC_MoveRelative



MC_MoveRelative starts a relative positioning procedure based on the current set position and monitors the axis movement over the whole travel path. The Done output is set once the target position has been reached. Otherwise, the output *CommandAborted* or, in case of an error, the output *Error* is set.

Motion commands can be applied to coupled slave axes, if this option was explicitly activated in the axis parameters. A motion command such as *MC_MoveAbsolute* then automatically leads to decoupling of the axis, after which the command is executed. In this case the only available buffer mode is *Aborting*.

Inputs

```
VAR_INPUT
Execute      : BOOL;
Distance     : LREAL;
Velocity     : LREAL;
Acceleration : LREAL;
Deceleration : LREAL;
Jerk         : LREAL;
BufferMode   : MC_BufferMode;
Options      : ST_MoveOptions;
END_VAR
```

Execute	The command is executed with a rising edge at *Execute* input.
Distance	Relative distance to be used for positioning.
Velocity	Maximum travel velocity (>0).
Acceleration	Acceleration (≥0). If the value is 0, the standard acceleration from the axis configuration in the System Manager is used.
Deceleration	Deceleration (≥0). If the value is 0, the standard deceleration from the axis configuration in the System Manager is used.
Jerk	Jerk (≥0). If the value is 0, the standard jerk from the axis configuration in the System Manager is used.
BufferMode	The BufferMode [▶ 103] is analyzed, if the axis is already executing another command. The running command can be aborted, or the new command becomes active after the running command. The transition condition from the current to the next command is also defined by the BufferMode. If the command is applied to a coupled slave axis, only the buffer mode *Aborting* is possible. To use the BufferMode, a second function block is always necessary. It is not possible to trigger a move function block with new parameters while it is active.

Options	The data structure Options includes additional, rarely required parameters. The input can normally remain open.

General rules for MC function blocks [▶ 14]

Outputs

```
VAR_OUTPUT
Done            : BOOL;
Busy            : BOOL;
Active          : BOOL;
CommandAborted : BOOL;
Error           : BOOL;
ErrorID         : UDINT;
END_VAR
```

Done	The *Done* output becomes TRUE once the target position was reached.
Busy	The *Busy* output becomes TRUE when the command is started with *Execute* and remains TRUE as long as the movement command is processed. If *Busy* becomes FALSE again, the function block is ready for a new order. At the same time one of the outputs, *Done*, *CommandAborted* or *Error*, is set.
Active	Active indicates that the command is executed. If the command was buffered, it becomes active once a running command is completed.
CommandAborted	Becomes TRUE, if the command could not be fully executed. The axis may have been stopped, or the running command may have been followed by a further Move command.
Error	Becomes TRUE if an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number.

General rules for MC function blocks [▶ 14]

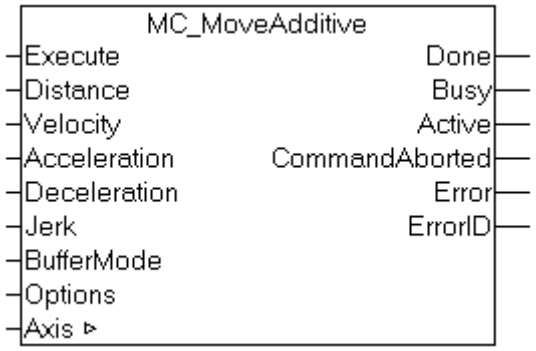
Inputs/outputs

```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Axis data structure

The axis data structure of type AXIS_REF [▶ 101] addresses an axis uniquely within the system. Among other parameters it contains the current axis status, including position, velocity or error status.

7.1.3 MC_MoveAdditive



MC_MoveAdditive starts relative positioning procedure based on the last target position instruction, irrespective of whether this was reached. The Done output is set once the target position has been reached. Otherwise the *CommandAborted* or, on error, the *Error* output is set.

If no last target position is known or the axis is moving continuously, the movement is executed based on the current set position for the axis.

Inputs

```
VAR_INPUT
Execute      : BOOL;
Distance     : LREAL;
Velocity     : LREAL;
Acceleration : LREAL;
Deceleration : LREAL;
Jerk         : LREAL;
BufferMode   : MC_BufferMode;
Options      : ST_MoveOptions;
END_VAR
```

Execute	The command is executed with a rising edge at input *Execute*.
Distance	Relative distance to be used for positioning.
Velocity	Maximum travel velocity (>0).
Acceleration	Acceleration (≥0). If the value is 0, the standard acceleration from the axis configuration in the System Manager is used.
Deceleration	Deceleration (≥0). If the value is 0, the standard deceleration from the axis configuration in the System Manager is used.
Jerk	Jerk (≥0). If the value is 0, the standard jerk from the axis configuration in the System Manager is used.
BufferMode	The BufferMode [▶ 103] is analyzed, if the axis is already executing another command. The running command can be aborted, or the new command becomes active after the running command. The BufferMode also determines the transition condition from the current to the next command. A second function block is required to use the buffer mode. It is not possible to trigger a move block with new parameters while it is active.
Options	The data structure option includes additional, rarely required parameters. The input can normally remain open.

General rules for MC function blocks [▶ 14]

Outputs

```
VAR_OUTPUT
Done            : BOOL;
Busy            : BOOL;
Active          : BOOL;
CommandAborted : BOOL;
Error           : BOOL;
ErrorID         : UDINT;
END_VAR
```

Done	The *Done* output becomes TRUE once the target position was reached.
Busy	The *Busy* output becomes TRUE when the command is started with *Execute* and remains TRUE as long as the movement command is processed. If *Busy* becomes FALSE again, the function block is ready for a new job. At the same time one of the outputs, *Done*, *CommandAborted* or *Error*, is set.
Active	Active indicates that the command is executed If the command was queued, it becomes active once a running command is completed.
CommandAborted	Becomes TRUE, if the command could not be fully executed. The axis may have been stopped, or the running command may have been followed by a further Move command.
Error	Becomes TRUE if an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number.

General rules for MC function blocks [▶ 14]

Inputs/outputs

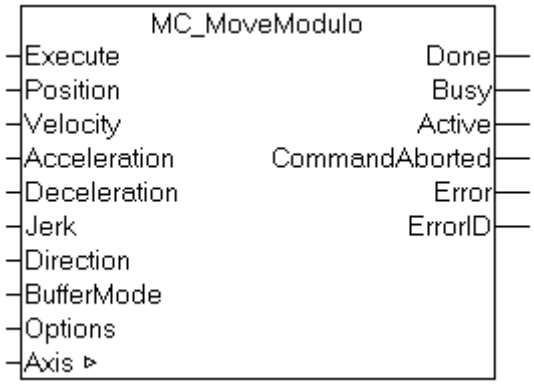
```
VAR_IN_OUT
Axis    : AXIS_REF;
END_VAR
```

Axis	Axis data structure

The axis data structure of type AXIS_REF [▶ 101] addresses an axis uniquely within the system. Among other parameters it contains the current axis status, including position, velocity or error status.

MC_MoveAdditive is not implemented for fast/slow axes.

7.1.4 MC_MoveModulo



The function block *MC_MoveModulo* carries out a positioning referenced to the modulo position of an axis. The basis for a modulo rotation is the adjustable axis parameter *modulo factor* (e.g. 360°). A distinction is made between three possible start types, depending on the *Direction* input.

- Positioning in *positive direction*
- Positioning in *negative direction*
- Positioning *along shortest path*

Motion commands can be applied to coupled slave axes, if this option was explicitly activated in the axis parameters. A motion command such as *MC_MoveModulo* then automatically leads to decoupling of the axis, after which the command is executed. In this case the only available buffer mode is *Aborting*.

Starting an axis from standstill

If an axis is started from standstill with *MC_MoveModulo* , it is possible to specify positions greater than or equal to 360°, in order to perform additional full turns. The same applies to a start with the *BufferModeMC_Buffered*.

Starting an axis during motion

If an axis is already in motion, certain special considerations apply. The direction of movement cannot be reversed by MC_MoveModulo, i.e. the target can only be reached in the current direction. The user is not able to specify the number of additional turns. The system automatically calculates how the axis can be moved to the target position on the shortest possible path.

The error output must be analyzed, because under certain conditions an oriented stop is not possible. For example, a standard stop may have been triggered just before, or an oriented stop would cause an active software limit switch to be exceeded. For all fault conditions, the axis is stopped safely, but it may subsequently not be at the required oriented position.

Special cases

Special attention must be paid to the behavior when one or more complete modulo rotations are requested. If the axis is located at an exact set position, such as 90 degrees, and if positioning to 90 degrees is required, no movement is carried out. If required to turn 450 degrees in a positive direction, it will perform just one rotation. The behavior can be different following an axis reset, because the reset will cause the current actual position to be adopted as the set position. The axis will then no longer be exactly at 90 degrees, but will be a little under or over. These cases will give rise either to a minimum positioning to 90 degrees, or on the other hand a complete rotation. For further details please refer to the Commentary [▶ 66] section.

Depending on the particular case, it may be more effective for complete modulo rotations to calculate the desired target position on the basis of the current absolute position, and then to position using the function block MC_MoveAbsolute [▶ 58].

Modulo positioning and absolute positioning are available for all axes, irrespective of the *Modulo* setting in the TwinCAT System Manager. For each axis, the current absolute position *SetPos* can be read from the cyclic axis interface data type NCTOPLC_AXIS_REF [▶ 102].

Important: Further information on modulo movements [▶ 66]

Inputs

```
VAR_INPUT
Execute : BOOL;
Position : LREAL;
Velocity : LREAL;
Acceleration : LREAL;
Deceleration : LREAL;
Jerk : LREAL;
Direction : MC_Direction;
BufferMode : MC_BufferMode;
Options : ST_MoveOptions;
END_VAR
```

MC_BufferMode [▶ 103] MC_Direction [▶ 105]

Execute	The command is executed with a rising edge at *Execute* input.
Position	Modulo target position to be used for positioning. If the axis is started from standstill, positions greater than 360° result in additional turns. Negative positions are not permitted.
Velocity	Maximum travel velocity (>0).
Acceleration	Acceleration (≥0). If the value is 0, the standard acceleration from the axis configuration in the System Manager is used.
Deceleration	Deceleration (≥0). If the value is 0, the standard deceleration from the axis configuration in the System Manager is used.
Jerk	Jerk (≥0). If the value is 0, the standard jerk from the axis configuration in the System Manager is used.
Direction	Positive or negative direction of travel of type MC_Direction [▶ 105]. If the axis is started during a motion, the direction may not be reversed.
BufferMode	The BufferMode [▶ 103] is analyzed, if the axis is already executing another command. The running command can be aborted, or the new command becomes active after the running command. The transition condition from the current to the next command is also defined by the BufferMode. To use the BufferMode, a second function block is always necessary. It is not possible to trigger a move function block with new parameters while it is active.
Options	The data structure Options includes additional, rarely required parameters. The input can normally remain open.

General rules for MC function blocks [▶ 14]

Outputs

```
VAR_OUTPUT
Done : BOOL;
Busy : BOOL;
```

```
Active : BOOL;
CommandAborted : BOOL;
Error : BOOL;
ErrorID : UDINT;
END_VAR
```

Done	The *Done* output becomes TRUE once the target position was reached.
Busy	The *Busy* output becomes TRUE when the command is started with *Execute* and remains TRUE as long as the movement command is processed. If *Busy* becomes FALSE again, the function block is ready for a new order. At the same time one of the outputs, *Done*, *CommandAborted* or *Error*, is set.
Active	Active indicates that the command is executed. If the command was buffered, it becomes active once a running command is completed.
CommandAborted	Becomes TRUE, if the command could not be fully executed. The axis may have been stopped, or the running command may have been followed by a further Move command.
Error	Becomes TRUE if an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number.

General rules for MC function blocks [▶ 14]

Inputs/outputs

```
VAR_IN_OUT
Axis : AXIS_REF;
END_VAR
```

AXIS_REF [▶ 101]

Axis	Axis data structure

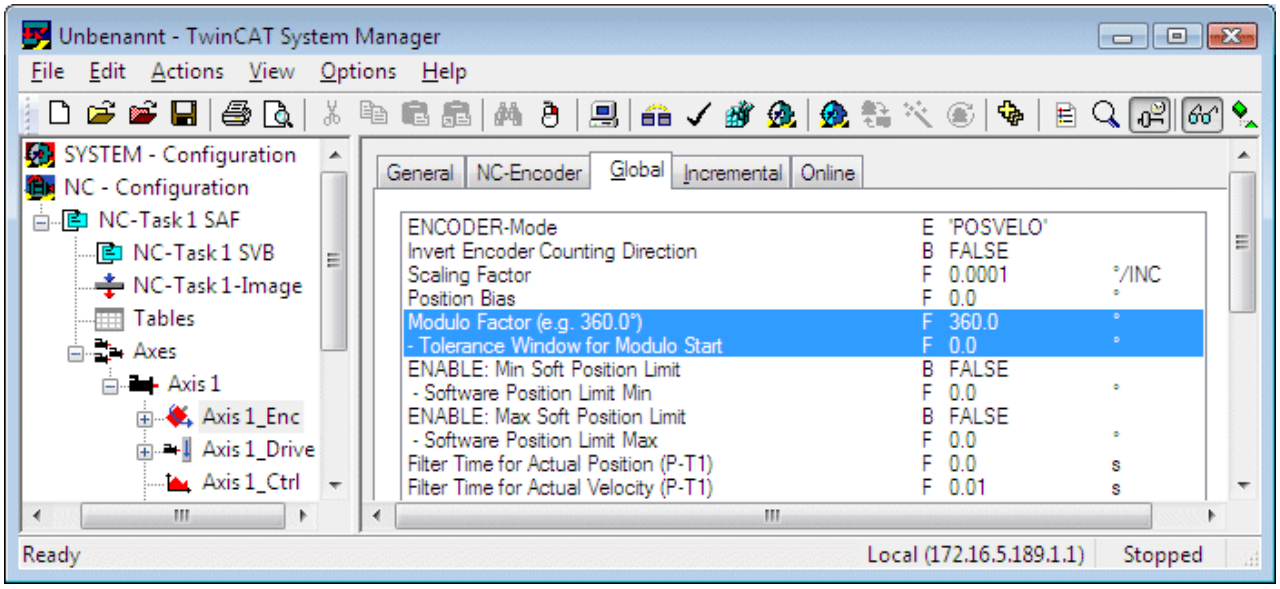
The axis data structure of type AXIS_REF [▶ 101] addresses an axis unambiguously within the system. Among other parameters it contains the current axis status, including position, velocity or error state.

7.1.5 Notes on modulo positioning

Modulo positioning (MC_MoveModulo [▶ 64]) is possible irrespective of the axis type. If may be used both for linear or rotary axes, because TwinCAT makes no distinction between these types. A modulo axis has a consecutive absolute position in the range ±∞. The modulo position of the axis is simply a piece of additional information about the absolute axis position. Modulo positioning represents the required target position in a different way. Unlike absolute positioning, where the user specifies the target unambiguously, modulo positioning has potential pitfalls, because the required target position may be interpreted in different ways.

Settings in the TwinCAT System Manager

Modulo positioning refers to a *modulo period* that can be set in the TwinCAT System Manager. The examples on this page assume a rotary axis with a *modulo period* of 360 degrees.



The *modulo tolerance window* defines a position window around the current modulo set position of the axis. The window width is twice the specified value (set position ± tolerance value). A detailed description of the tolerance window is provided below.

Special features of axis resets

Axis positioning always refers to the set position. The set position of an axis is normally the target position of the last travel command. An axis reset (MC_Reset [▶ 20], controller activation with MC_Power [▶ 19]) can lead to a set position that is different from that expected by the user, because in this case the current actual position is used as the set position. The axis reset will reset any following error that may have occurred. If this possibility is not considered, subsequent positioning may lead to unexpected behavior.

Example: An axis is positioned to 90°, with the result that subsequently the set position of the axis is exactly 90°. A further modulo travel command to 450° in *positive direction* results in a full turn, with the subsequent modulo position of the axis of once again being exactly 90°. If an axis reset is carried out at this stage, the set position may happen to be somewhat smaller or greater. The new value depends on the actual value of the axis at the time of the reset. However, the next travel command will lead to different results. If the set position is slightly less than 90°, a new travel command to 90° in *positive direction* only leads to minimum motion. The deviation created by the reset is compensated, and the subsequent set position is once again exactly 90°. However, if the set position after the axis reset is slightly more than 90°, the same travel command leads to a full turn to reach the exact set position of 90°. This problem occurs if complete turns by 360° or multiples of 360° were initiated. For positioning to an angle that is significantly different from the current modulo position, the travel command is unambiguous.

To solve the problem, a *modulo tolerance window* can be parameterized in the TwinCAT system manager. This ensures that small deviations from the position that are within the window do not lead to different axis behavior. If, for example, a window of 1° is parameterized, in the case described above the axis will behave identically, as long the set position is between 89° and 91°. If the set position exceeds 90° by less than 1°, the axis is re-positioned in *positive direction* at a modulo start. In both cases, a target position of 90° therefore leads to minimum movement to exactly 90°. A target position of 450° leads to a full turn in both cases.

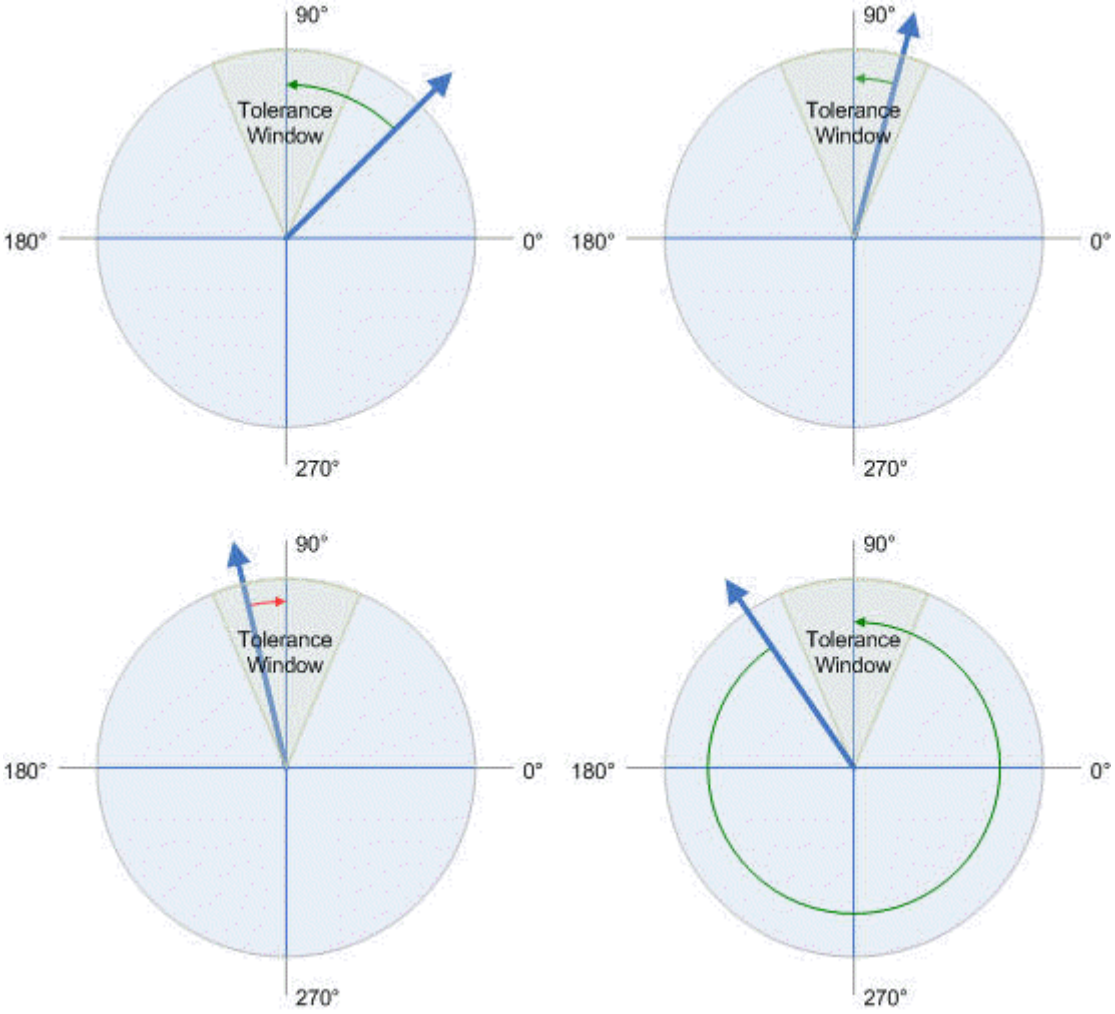


Figure: Effect of the modulo tolerance window - modulo target position 90° in positive direction

For values that are within the window range, the modulo tolerance window can therefore lead to movements against the specified direction. For small windows this is usually not a problem, because system deviations between set and actual position are compensated in both directions. This means that the tolerance window may also be used for axes that may only be moved in one direction due to their construction.

Modulo positioning by less than one turn

Modulo positioning from a starting position to a non-identical target position is unambiguous and requires no special consideration. A modulo target position in the range [0 ≤; position < 360] reaches the required target in less than one whole turn. No motion occurs if target position and starting position are identical. Target positions of more than 360 degrees lead to one or more full turns before the axis travels to the required target position.

For a movement from 270° to 0°, a modulo target position of 0° (not 360°) should therefore be specified, because 360 is outside the basic range and would lead to an additional turn.

For modulo positioning, a distinction is made between three different directions, i.e. *positive direction*, *negative direction* and *along shortest path* (MC_Direction [▶ 105]). For positioning along the shortest path, target positions of more than 360° are not sensible, because the movement towards the target is always direct. In contrast to positive or negative direction, it is therefore not possible to carry out several turns before the axis moves to the target.

Important: For modulo positioning with start type *along shortest path* , only modulo target positions within the basic period (e.g. less than 360°) are permitted, otherwise an error is returned.

The following table shows some positioning examples:

Direction (modulo start type)	Absolute start position	Modulo target position	Relative travel path	Absolute end position	Modulo end position
positive direction	90.00	0.00	270.00	360.00	0.00
positive direction	90.00	360.00	630.00	720.00	0.00
positive direction	90.00	720.00	990.00	1080.00	0.00
negative direction	90.00	0.00	-90.00	0.00	0.00
negative direction	90.00	360.00	-450.00	-360.00	0.00
negative direction	90.00	720.00	-810.00	-720.00	0.00
along shortest path	90.00	0.00	-90.00	0.00	0.00

Modulo positioning with full turns

In principle, modulo positioning by one or full turns are no different than positioning to an angle that differs from the starting position. No motion occurs if target position and starting position are identical. For a full turn, 360° has to be added to the starting position.

The reset behavior described above shows that positioning with full turns requires particular attention. The following table shows positioning examples for a starting position of approximately 90°. The modulo tolerance window (TW) is set to 1°. Special cases for which the starting position is outside this window are identified.

Direction (modulo start type)	Absolute start position	Modulo target position	Relative travel path	Absolute end position	Modulo end position	Note
positive direction	90.00	90.00	0.00	90.00	90.00	
positive direction	90.90	90.00	-0.90	90.00	90.00	
positive direction	91.10	90.00	358.90	450.00	90.00	outside TF
positive direction	89.10	90.00	0.90	90.00	90.00	
positive direction	88.90	90.00	1.10	90.00	90.00	outside TF
positive direction	90.00	450.00	360.00	450.00	90.00	
positive direction	90.90	450.00	359.10	450.00	90.00	
positive direction	91.10	450.00	718.90	810.00	90.00	outside TF
positive direction	89.10	450.00	360.90	450.00	90.00	
positive direction	88.90	450.00	361.10	450.00	90.00	outside TF
positive direction	90.00	810.00	720.00	810.00	90.00	

positive direction	90.90	810.00	719.10	810.00	90.00
positive direction	91.10	810.00	1078.90	1170.00	90.00 outside TF
positive direction	89.10	810.00	720.90	810.00	90.00
positive direction	88.90	810.00	721.10	810.00	90.00 outside TF
negative direction	90.00	90.00	0.00	90.00	90.00
negative direction	90.90	90.00	-0.90	90.00	90.00
negative direction	91.10	90.00	-1.10	90.00	90.00 outside TF
negative direction	89.10	90.00	0.90	90.00	90.00
negative direction	88.90	90.00	-358.90	-270.00	90.00 outside TF
negative direction	90.00	450.00	-360.00	-270.00	90.00
negative direction	90.90	450.00	-360.90	-270.00	90.00
negative direction	91.10	450.00	-361.10	-270.00	90.00 outside TF
negative direction	89.10	450.00	-359.10	-270.00	90.00
negative direction	88.90	450.00	-718.90	-630.00	90.00 outside TF
negative direction	90.00	810.00	-720.00	-630.00	90.00
negative direction	90.90	810.00	-720.90	-630.00	90.00
negative direction	91.10	810.00	-721.10	-630.00	90.00 outside TF
negative direction	89.10	810.00	-719.10	-630.00	90.00
negative direction	88.90	810.00	-1078.90	-990.00	90.00 outside TF

Modulo calculations within the PLC program

In TwinCAT NC, all axis positioning tasks are executed based on the *set position*. The current actual position is only used for control purposes. If a PLC program is to calculate a new target position based on the current position, the current set position of the axis has to be used in the calculation (Axis.NcToPlc.ModuloSetPos and Axis.NcToPlc.ModuloSetTurns).

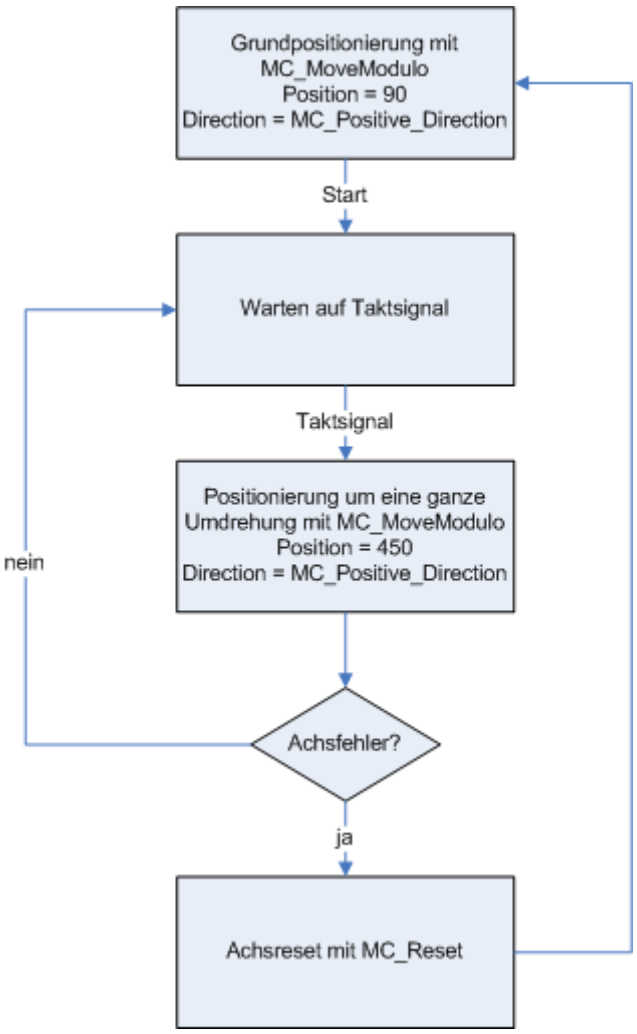
It is not recommended to perform order calculations on basis of the modulo actual position available in the cyclic axis interface (*ModuloActPos* and *ModuloActTurns*). Due to the larger or smaller control deviation of the axis, errors can occur in the programmed sequence, such as unwanted rotations.

Application example

Within a system, a rotational axis carries out an operation. The starting position for each operation is 90°, and with each cycle the axis is to be positioned by 360° in positive direction. Reverse positioning is not permitted for mechanical reasons. Small reverse positioning is acceptable as part of position control movements.

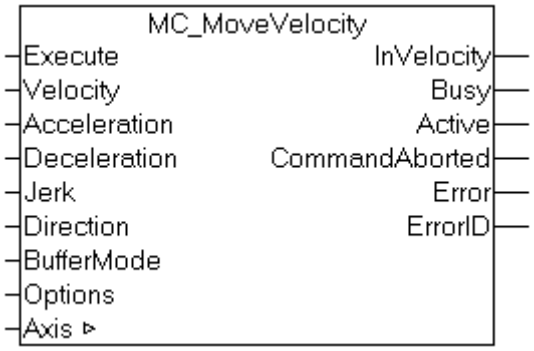
The modulo tolerance window is set to 1.5° in the System Manager. This ensures that undesirable axis turns after an axis reset are avoided. Since the axis may only be positioned in positive direction, the command MC_MoveModulo [▶ 64] with modulo start type *positive direction (MC_Positive_Direction)* is used. The modulo target position is specified as 450°, since the original orientation is to be reached again after a full turn by 360°. A modulo target position of 90° would not lead to any motion.

The process starts with a basic positioning movement (MC_MoveModulo [▶ 64]) to ensure that the starting position is accurate. The step sequence then changes into an execution cycle. In the event of a fault, the axis is reset with MC_Reset [▶ 20] and subsequently (at the start of the step sequence) moved to its valid starting position. In this case, 90° is specified as the target position to enable this position to be reached as quickly as possible. No motion occurs if the axis is already at the starting position.



Alternatively, the reset step may be carried out at the start of the step sequence, so that the axis is initialized at the start of the process.

7.1.6 MC_MoveVelocity



MC_MoveVelocity starts a continuous movement with specified velocity and direction. The movement can be stopped through a Stop command.

The *InVelocity* output is set once the constant velocity is reached. Once constant velocity has been reached, the block function is complete, and no further monitoring of the movement takes place. If the command is aborted during the acceleration phase, the CommandAborted or, on error, the Error output is set.

Motion commands can be applied to coupled slave axes, if this option was explicitly activated in the axis parameters. A motion command such as *MC_MoveAbsolute* then automatically leads to decoupling of the axis, after which the command is executed. In this case the only available buffer mode is *Aborting*.

Inputs

```
VAR_INPUT
Execute : BOOL; (* B *)
Velocity : LREAL; (* E *)
Acceleration : LREAL; (* E *)
Deceleration : LREAL; (* E *)
Jerk : LREAL; (* E *)
Direction : MC_Direction := MC_Positive_Direction; (* E *)
BufferMode : MC_BufferMode; (* E *)
Options : ST_MoveOptions; (* V *)
END_VAR
```

MC_BufferMode [▶ 103] MC_Direction [▶ 105]

Execute	The command is executed with a rising edge at *Execute* input.
Velocity	Travel velocity (>0).
Acceleration	Acceleration (≥0). If the value is 0, the standard acceleration from the axis configuration in the System Manager is used.
Deceleration	Deceleration (≥0). If the value is 0, the standard deceleration from the axis configuration in the System Manager is used.
Jerk	Jerk (≥0). If the value is 0, the standard jerk from the axis configuration in the System Manager is used.
Direction	Positive or negative direction of travel of type MC_Direction [▶ 105].
BufferMode	The BufferMode [▶ 103] is analyzed, if the axis is already executing another command. The running command can be aborted, or the new command becomes active after the running command. The transition condition from the current to the next command is also defined by the BufferMode. If the command is applied to a coupled slave axis,

	only the buffer mode *Aborting* is possible. To use the BufferMode, a second function block is always necessary. It is not possible to trigger a move function block with new parameters while it is active.
Options	The data structure Options includes additional, rarely required parameters. The input can normally remain open.

General rules for MC function blocks [▶ 14]

Outputs

```
VAR_OUTPUT
InVelocity : BOOL; (* B *)
Busy : BOOL; (* E *)
Active : BOOL; (* E *)
CommandAborted : BOOL; (* E *)
Error : BOOL; (* B *)
ErrorID : UDINT; (* E *)
END_VAR
```

InVelocity	The output *InVelocity* becomes TRUE, as soon as the constant velocity is reached. It may switch back to FALSE, if the velocity differs. The function block remains *Busy* and *Active* until a new command is issued.
Busy	The *Busy* output becomes TRUE as soon as the command is started with *Execute* and remains TRUE as long as the function block is active. If *Busy* becomes FALSE again, the function block is ready for a new order. At the same time one of the outputs, *CommandAborted* or *Error*, is set.
Active	Active indicates that the command is executed. If the command was buffered, it becomes active once a running command is completed.
CommandAborted	Becomes TRUE, if the command could not be fully executed. The axis may have been stopped, or the running command may have been followed by a further Move command.
Error	Becomes TRUE if an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number.

General rules for MC function blocks [▶ 14]

Inputs/outputs

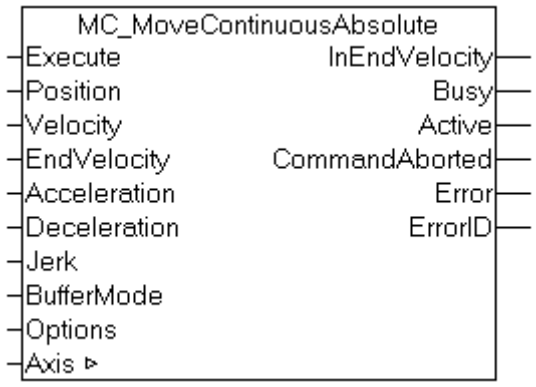
```
VAR_IN_OUT
Axis : AXIS_REF;
END_VAR
```

AXIS_REF [▶ 101]

Axis	Axis data structure

The axis data structure of type AXIS_REF [▶ 101] addresses an axis unambiguously within the system. Among other parameters it contains the current axis status, including position, velocity or error state.

7.1.7 MC_MoveContinuousAbsolute



MC_MoveContinuousAbsolute starts positioning to an absolute target position and monitors the axis movement over the whole travel path. At the target position a constant end velocity is reached, which is maintained. The *InEndVelocity* output is set once the target position was reached. Otherwise the CommandAborted or, on error, the Error output is set.

Once the target position has been reached, the block function is complete and the axis is no longer monitored.

Inputs

```
VAR_INPUT
Execute : BOOL;
Position : LREAL;
Velocity : LREAL;
EndVelocity : LREAL;
Acceleration : LREAL;
Deceleration : LREAL;
Jerk : LREAL;
BufferMode : MC_BufferMode;
Options : ST_MoveOptions;
END_VAR
```

MC_BufferMode [▶ 103]

Execute	The command is executed with a rising edge at input *Execute*.
Position	Absolute target position
Velocity	Maximum velocity for the movement to the target position (>0).
EndVelocity	End velocity to be maintained once the target position has been reached.
Acceleration	Acceleration (≥0). If the value is 0, the standard acceleration from the axis configuration in the System Manager is used.
Deceleration	Deceleration (≥0). If the value is 0, the standard deceleration from the axis configuration in the System Manager is used.
Jerk	Jerk (≥0). If the value is 0, the standard jerk from the axis configuration in the System Manager is used.
BufferMode	The BufferMode [▶ 103] is analyzed, if the axis is already executing another command. The running command can be aborted, or the new command becomes active after the running command. The BufferMode also determines the transition condition from the current to the next command.

	A second function block is required to use the buffer mode. It is not possible to trigger a move block with new parameters while it is active.
Options	The data structure option includes additional, rarely required parameters. The input can normally remain open.

General rules for MC function blocks [▶ 14]

Outputs

```
VAR_OUTPUT
InEndVelocity : BOOL;
Busy : BOOL;
Active : BOOL;
CommandAborted : BOOL;
Error : BOOL;
ErrorID : UDINT;
END_VAR
```

InEndVelocity	The InEndVelocity output becomes TRUE once the target position was reached. The function block remains *Busy* and *Active* until a new command is issued.
Busy	The *Busy* output becomes TRUE when the command is started with *Execute* and remains TRUE as long as the movement command is processed. If *Busy* becomes FALSE again, the function block is ready for a new order. At the same time one of the outputs, Done, *CommandAborted* or *Error*, is set.
Active	Active indicates that the command is executed. If the command was buffered, it becomes active once a running command is completed.
CommandAborted	Becomes TRUE, if the command could not be fully executed. The axis may have been stopped, or the running command may have been followed by a further Move command.
Error	Becomes TRUE if an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number.

General rules for MC function blocks [▶ 14]

Inputs/outputs

```
VAR_IN_OUT
Axis : AXIS_REF;
END_VAR
```

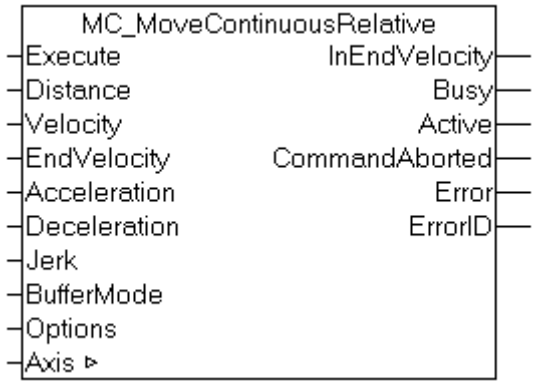
AXIS_REF [▶ 101]

Axis	Axis data structure

The axis data structure of type AXIS_REF [▶ 101] addresses an axis uniquely within the system. Among other parameters it contains the current axis status, including position, velocity or error status.

MC_MoveContinuousAbsolute is not implemented for fast/slow axes.

7.1.8 MC_MoveContinuousRelative



MC_MoveContinuousRelative starts positioning by a relative distance and monitors the axis movement over the whole travel path. At the target position a constant end velocity is reached, which is maintained. The *InEndVelocity* output is set once the target position was reached. Otherwise the CommandAborted or, on error, the Error output is set.

Once the target position has been reached, the block function is complete and the axis is no longer monitored.

Inputs

```
VAR_INPUT
Execute : BOOL;
Distance : LREAL;
Velocity : LREAL;
EndVelocity : LREAL;
Acceleration : LREAL;
Deceleration : LREAL;
Jerk : LREAL;
BufferMode : MC_BufferMode;
Options : ST_MoveOptions;
END_VAR
```

MC_BufferMode [▶ 103]

Execute	The command is executed with a rising edge at input *Execute*.
Distance	Relative distance to be used for positioning.
Velocity	Maximum velocity for the movement over the *distance* (>0).
EndVelocity	End velocity to be maintained after the relative *distance*
Acceleration	Acceleration (≥0). If the value is 0, the standard acceleration from the axis configuration in the System Manager is used.
Deceleration	Deceleration (≥0). If the value is 0, the standard deceleration from the axis configuration in the System Manager is used.
Jerk	Jerk (≥0). If the value is 0, the standard jerk from the axis configuration in the System Manager is used.
BufferMode	The BufferMode [▶ 103] is analyzed, if the axis is already executing another command. The running command can be aborted, or the new command becomes active after the running command. The BufferMode also determines the transition condition from the current to the next command.

	A second function block is required to use the buffer mode. It is not possible to trigger a move block with new parameters while it is active.
Options	The data structure option includes additional, rarely required parameters. The input can normally remain open.

General rules for MC function blocks [▶ 14]

Outputs

```
VAR_OUTPUT
InEndVelocity : BOOL;
Busy : BOOL;
Active : BOOL;
CommandAborted : BOOL;
Error : BOOL;
ErrorID : UDINT;
END_VAR
```

InEndVelocity	The InEndVelocity output becomes TRUE once the target position was reached. The function block remains *Busy* and *Active* until a new command is issued.
Busy	The *Busy* output becomes TRUE when the command is started with *Execute* and remains TRUE as long as the movement command is processed. If *Busy* becomes FALSE again, the function block is ready for a new order. At the same time one of the outputs, Done, *CommandAborted* or *Error*, is set.
Active	Active indicates that the command is executed. If the command was buffered, it becomes active once a running command is completed.
CommandAborted	Becomes TRUE, if the command could not be fully executed. The axis may have been stopped, or the running command may have been followed by a further Move command.
Error	Becomes TRUE if an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number.

General rules for MC function blocks [▶ 14]

Inputs/outputs

```
VAR_IN_OUT
Axis : AXIS_REF;
END_VAR
```

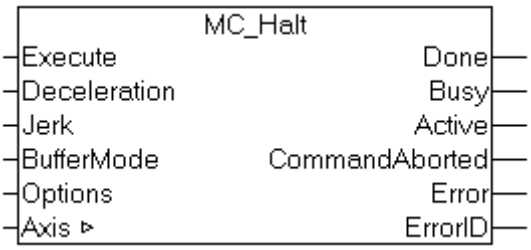
AXIS_REF [▶ 101]

Axis	Axis data structure

The axis data structure of type AXIS_REF [▶ 101] addresses an axis uniquely within the system. Among other parameters it contains the current axis status, including position, velocity or error status.

MC_MoveContinuousRelative is not implemented for fast/slow axes.

7.1.9 MC_Halt



MC_Halt stops an axis with a defined braking ramp.

In contrast to MC_Stop [▶ 80], the axis is not locked against further movement commands. The axis can therefore be restarted through a further command during the braking ramp or after it has come to a halt.

Travel commands can be applied to coupled slave axes, if this option was explicitly activated in the axis parameters. A motion command such as *MC_Halt* then automatically leads to uncoupling of the axis, after which the command is executed. In this case only *Buffer-ModeAborting* is possible.

Inputs

```
VAR_INPUT
Execute : BOOL;
Deceleration : LREAL;
Jerk : LREAL;
BufferMode : MC_BufferMode;
Options : ST_MoveOptions;
END_VAR
```

MC_BufferMode [▶ 103]

Execute	The command is executed with a rising edge at input *Execute*.
Deceleration	Deceleration (≥0). If the value is 0, the deceleration parameterized with the last Move command is used. For safety reasons *MC_Halt* and MC_Stop [▶ 80] cannot be executed with weaker dynamics than the currently active travel command. The parameterization is adjusted automatically, if necessary.
Jerk	Jerk (≥0). If the value is 0, the jerk parameterized with the last Move command is used. For safety reasons *MC_Halt* and MC_Stop [▶ 80] cannot be executed with weaker dynamics than the currently active travel command. The parameterization is adjusted automatically, if necessary.
BufferMode	The BufferMode [▶ 103] is analyzed, if the axis is already executing another command. The running command can be aborted, or the new command becomes active after the running command. The BufferMode also determines the transition condition from the current to the next command. If the command is applied to a coupled slave axis used, the only available buffer mode is *Aborting*. Special characteristics of *MC_Halt*: The *MC_buffer* mode has no effect, if the command is executed when the system is at a standstill. The blending modes *MC_BlendingNext* and *MC_BlendingLow* do not change the last target position, although they can

	result in a change in dynamics (*deceleration*) of the stop ramp. The modes *MC_BlendingPrevious* and *MC_BlendingHigh* extend the travel to the original target position. The stop ramp is only initiated when this position is reached (defined braking point).
Options	Currently not implemented - The data structure option includes additional, rarely required parameters. The input can normally remain open.

General rules for MC function blocks [▶ 14]

Outputs

```
VAR_OUTPUT
Done : BOOL;
Busy : BOOL;
Active : BOOL;
CommandAborted : BOOL;
Error : BOOL;
ErrorID : UDINT;
END_VAR
```

Done	The *Done* output becomes TRUE, if the axis was stopped and has come to a standstill.
Busy	The *Busy* output becomes TRUE when the command is started with *Execute* and remains TRUE as long as the command is processed. If *Busy* becomes FALSE again, the function block is ready for a new job. At the same time one of the outputs, *Done*, *CommandAborted* or *Error*, is set.
Active	Active indicates that the command is executed If the command was queued, it becomes active once a running command is completed.
CommandAborted	Becomes TRUE, if the command could not be fully executed. The running command may have been followed by a Move command.
Error	Becomes TRUE if an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number.

General rules for MC function blocks [▶ 14]

Inputs/outputs

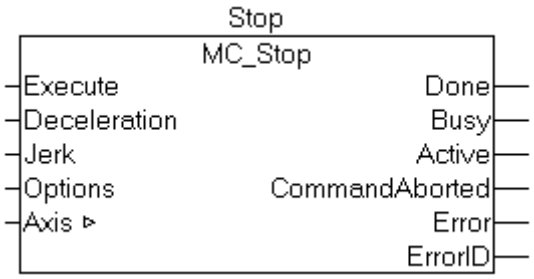
```
VAR_IN_OUT
Axis : AXIS_REF;
END_VAR
```

AXIS_REF [▶ 101]

Axis	Axis data structure

The axis data structure of type AXIS_REF [▶ 101] addresses an axis uniquely within the system. Among other parameters it contains the current axis status, including position, velocity or error status.

7.1.10 MC_Stop



MC_Stop stops an axis with a defined deceleration ramp and locks it against other motion commands. The function block is therefore suitable for stops in special situations, in which further axis movements are to be prevented.

NOTICE

At the same time the axis is blocked for other motion commands. The axis can only be restarted once the Execute signal has been set to FALSE after the axis has stopped. A few cycles are required to release the axis after a falling edge of Execute. During this phase the Busy output remains TRUE, and the function block has to be called until Busy becomes FALSE.

NOTICE

The locking of the axis is canceled with an MC_Reset.

Alternatively, the axis can be stopped with MC_Halt [▶ 78] without locking. *MC_Halt* is preferable for normal movements.

Motion commands can be applied to coupled slave axes, if this option was explicitly activated in the axis parameters. A motion command such as *MC_Stop* then automatically leads to decoupling of the axis, after which the command is executed.

Inputs

```
VAR_INPUT
Execute : BOOL;
Deceleration : LREAL;
Jerk : LREAL;
Options : ST_MoveOptions;
END_VAR
```

Execute	The command is executed with a rising edge at input *Execute*. The axis is locked during the stop. The axis can only be restarted once the Execute signal has been set to FALSE after the axis has stopped.
Deceleration	Deceleration (≥0). If the value is 0, the deceleration parameterized with the last move command takes effect. For safety reasons *MC_Stop* and *MC_Halt* cannot be executed with weaker dynamics than the currently active motion command. The parameterization is adjusted automatically, if necessary.
Jerk	Jerk (≥0). If the value is 0, the jerk parameterized with the last Move command takes effect. For safety reasons *MC_Stop* and *MC_Halt* cannot be executed with weaker dynamics than the currently active motion command. The parameterization is adjusted automatically, if necessary.

Options	Currently not implemented - The data structure Options includes additional, rarely required parameters. The input can normally remain open.

General rules for MC function blocks [▶ 14]

Outputs

```
VAR_OUTPUT
Done : BOOL;
Busy : BOOL;
Active : BOOL;
CommandAborted : BOOL;
Error : BOOL;
ErrorID : UDINT;
END_VAR
```

Done	The *Done* output becomes TRUE, if the axis was stopped and has come to a standstill.
Busy	The *Busy* output becomes TRUE when the command is started with *Execute* and remains TRUE as long as the command is processed. If *Busy* becomes FALSE again, the function block is ready for a new order. **Notice Busy remains TRUE as long as the axis is locked. The axis is only unlocked and Busy becomes FALSE when Execute is set to FALSE.**
Active	Active indicates that the function block is controlling the axis. **Notice Active remains TRUE as long as the axis is locked. The axis is only unlocked and Active becomes FALSE when Execute is set to FALSE.**
CommandAborted	Becomes TRUE, if the command could not be fully executed.
Error	Becomes TRUE if an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number.

General rules for MC function blocks [▶ 14]

Inputs/outputs

```
VAR_IN_OUT
Axis : AXIS_REF;
END_VAR
```

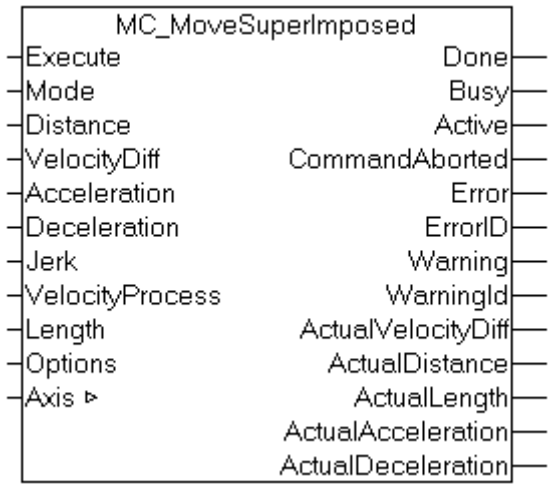
AXIS_REF [▶ 101]

Axis	Axis data structure

The axis data structure of type AXIS_REF [▶ 101] addresses an axis unambiguously within the system. Among other parameters it contains the current axis status, including position, velocity or error state.

7.2 Superposition

7.2.1 MC_MoveSuperimposed



MC_MoveSuperimposed starts a relative superimposed movement while the axis is already moving. The current movement is not interrupted. The *Done* output is set once the superimposed movement is completed. The original subordinate movement may continue to be active and is monitored by the associated Move function block.

The superposition function becomes clear, if one considers two axes moving at the same speed. If one of the axes is superimposed by *MC_MoveSuperimposed*, it will precede or follow the other axis as determined by the *Distance* parameter. Once the superimposed movement is completed, the *Distance* between the two axes is maintained.

MC_MoveSuperimposed can be applied to single axes, master axes or slave axes. For a slave axis the superimposed movement only affects the slave axis. If the function is used for a master axis, the slave will follow the superimposed movement of the due master due to the axis coupling.

Since *MC_MoveSuperimposed* executes a relative superimposed movement, the target position for the subordinate travel command changes by *Distance*.

The superimposed movement depends on the position of the main movement. This means that a velocity change of the main movement also results in a velocity change in the superimposed movement, and that the superimposed movement is inactive if the main movement stops. The *Options* parameter can be used to specify whether the superimposed movement is to be aborted or continued if the main movement stops.

Application examples for MC_MoveSuperimposed [▶ 84]

Inputs

```
VAR_INPUT
Execute : BOOL; (* B *)
Mode : E_SuperpositionMode;
Distance : LREAL; (* B *)
VelocityDiff : LREAL; (* E *)
Acceleration : LREAL; (* E *)
Deceleration : LREAL; (* E *)
Jerk : LREAL; (* E *)
VelocityProcess : LREAL; (* V *)
Length : LREAL; (* V *)
Options : ST_SuperpositionOptions; (* V *)
END_VAR
```

ST_SuperpositionOptions [▶ 107] E_SuperpositionMode [▶ 106]

Execute	The command is executed with a rising edge at input *Execute*.

Mode	Mode [▶ 106] determines the type of the superimposed motion.	
Distance	Relative distance to catch up. A positive value means increase in velocity by an amount required to cover the additional distance, compared with the unaffected movement. A negative value results in braking and falling back by this distance.	
VelocityDiff	Maximum velocity difference to the current velocity (basic velocity) of the axis (>0). For this parameter a distinction may have to be made, depending on the superimposition direction (acceleration or deceleration). If, for example, a direction reversal is not permitted, the maximum available acceleration corresponds to the maximum velocity, and the maximum deceleration to stop. Therefore, there are two possible maximum values for VelocityDiff: 1. 1. Distance > 0 (axis accelerates) VelocityDiff = maximum speed - basic speed 2. Distance > 0 (axis decelerates) VelocityDiff = basic speed	
Acceleration	Acceleration (≥0). If the value is 0, the standard acceleration from the axis configuration in the System Manager is used.	
Deceleration	Deceleration (≥0). If the value is 0, the standard deceleration from the axis configuration in the System Manager is used.	
Jerk	Jerk (≥0). If the value is 0, the standard jerk from the axis configuration in the System Manager is used.	
VelocityProcess :	Mean process speed in the axis (>0).If the basic velocity during superposition is constant, the set axis velocity can be specified.	
Length	Distance over which the superimposed movement is available. The *Mode* parameter defines how this distance is interpreted.	
Options	The data structure option includes additional, rarely required parameters. The input can normally remain open.	
Options.	**AbortOption**	AbortOption defines the behavior when the subordinate movement stops. The superimposed movement can be aborted or continued later.

General rules for MC function blocks [▶ 14]

Outputs

```
VAR_OUTPUT
Done : BOOL;
Busy : BOOL;
Active : BOOL;
CommandAborted : BOOL;
Error : BOOL;
ErrorID : UDINT;
Warning : BOOL;
WarningID : UDINT;
ActualVelocityDiff : LREAL;
ActualDistance : LREAL;
ActualLength : LREAL;
ActualAcceleration : LREAL;
ActualDeceleration : LREAL;

END_VAR
```

Done	The *Done* output becomes TRUE, once the superimposed movement was completed successfully.
Busy	The *Busy* output becomes TRUE when the command is started with *Execute* and remains TRUE as long as the movement command is processed. If *Busy*

	becomes FALSE again, the function block is ready for a new job. At the same time one of the outputs, *Done*, *CommandAborted* or *Error*, is set.
Active	Active indicates that the command is executed
CommandAborted	Becomes TRUE, if the command was aborted by another command and could therefore not be completed.
Error	Becomes TRUE if an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number.
Warning	Warning becomes TRUE if the action cannot be executed completely.
WarningID	The block returns warning 4243_{hex} (16963) if the compensation was incomplete due to the parameterization (distance, velocity, etc.). In this case compensation is implemented as far as possible. The user has to decide whether to interpret this warning message within his application as a proper error or merely as a warning.
ActualVelocityDiff:	Actual velocity difference during the superimposed motion (ActualVelocityDiff ≤ VelocityDiff).
ActualDistance:	Actual superimposed distance. The block tries to reach the full *Distance* as specified. This distance may not be reached fully, depending on the parameterization (*VelocityDiff, Acceleration, Deceleration, Length, Mode*). In this case the maximum possible distance is superimposed. (*ActualDistance* ≤ *Distance*).
ActualLength	Actual travel during superimposed motion (*ActualLenght* ≤ *Length*).
ActualAcceleration	Actual acceleration of the superimposed movement (*ActualAcceleration*≤A*cceleration*).
ActualDeceleration	Actual deceleration of the superimposed movement (*ActualDeceleration*≤D*eceleration*).

General rules for MC function blocks [▶ 14]

Inputs/outputs

```
VAR_IN_OUT
Axis : AXIS_REF;
END_VAR
```

AXIS_REF [▶ 101]

Axis	Axis data structure

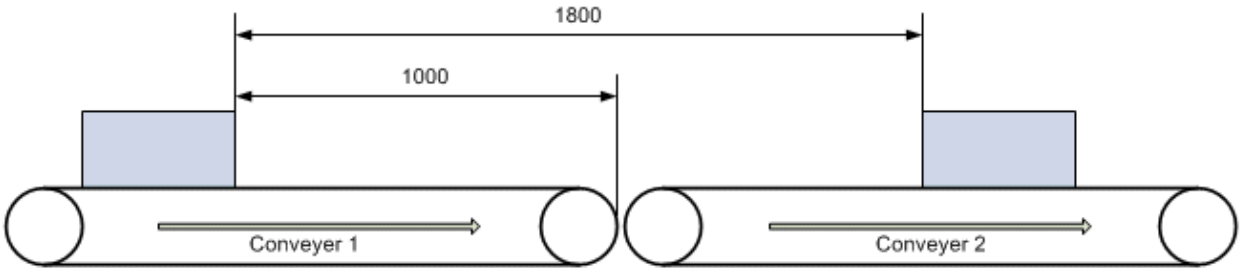
The axis data structure of type AXIS_REF [▶ 101] addresses an axis uniquely within the system. Among other parameters it contains the current axis status, including position, velocity or error status.

7.2.2 Application examples for MC_MoveSuperimposed

The function block MC_MoveSuperimposed [▶ 82] starts a superimposed movement on an axis that is already moving. For this superposition various applications are available that are described below.

Distance correction for products on a conveyor belt

A conveyor belt consists of individual segments, each driven by an axis. The conveyor belt is used for transporting packages, the spacing of which is to be corrected. To this end a conveying segment must briefly run faster or slower relative to a following segment.



The measured distance is 1800 mm and is to be reduced to 1500 mm. Conveyor belt 1 should be accelerated in order to reduce the distance. The correction must be completed by the time the end of belt 1 is reached in order to prevent the package being pushed onto the slower belt 2.

Since in this situation conveyer 1 has to be accelerated the drive system requires a velocity reserve, assumed to be 500 mm/s in this case. In practice this value can be determined from the difference between the maximum conveyor speed and the current set velocity.

For parameterization of function block MC_MoveSuperimposed [▶ 82] this means:

Distance = 1800 mm - 1500 mm = 300 mm (distance correction)

Length = 1000 mm (available distance up to the end of belt 1)

Mode = SUPERPOSITIONMODE_VELOREDUCTION_LIMITEDMOTION

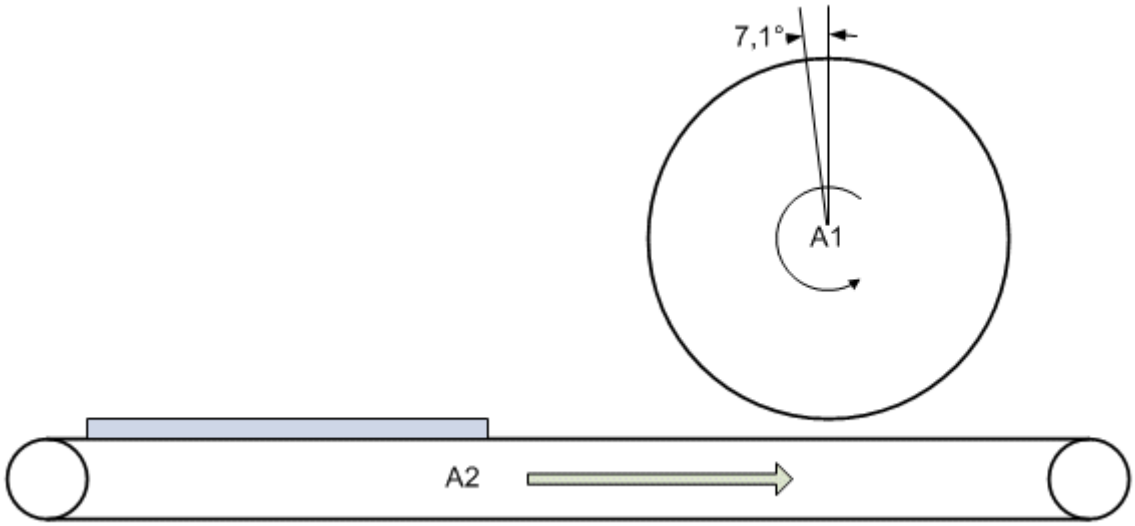
VelocityDiff = 500 mm/s

The mode defines that the distance *Length* up to the end of the conveyor belt is used for the correction and that the correction is completed at this point. The system uses the internally calculated velocity as degree of freedom. *VelocityDiff* therefore is the upper limit for the velocity change in this case.

Alternatively the correction could be achieved by decelerating belt 2. In this case *Distance* must be negative and the available correction distance *Length* is the distance between the right-hand package and the end of the belt. The maximum possible velocity change *VelocityDiff* corresponds to the current set velocity. Belt 2 could therefore be decelerated down to zero, if necessary.

Phase shift of a print roller

A print roller rotates with constant peripheral velocity at the same speed as conveyor belt on which a workpiece to be printed is transported. For synchronization with the workpiece the print roller is to be advanced by a certain angle (phase shift).



The phase shift can be implemented in two ways. The angle can be corrected as quickly as possible, resulting in a short-term strong increase in the velocity of the print roller. Alternatively a correction distance can be defined within which the correction can occur, e.g. a complete roller revolution. This leads to the following possible parameterizations for function block MC_MoveSuperimposed [▶ 82]:

1. Fast correction:

Distance = 7.1°

Length = 360° (maximum possible correction distance)

Mode = SUPERPOSITIONMODE_LENGTHREDUCTION_LIMITEDMOTION

VelocityDiff = 30°/s (velocity reserve)

The *mode* specifies that the correction distance should be as short as possible. The stated value for *Length* therefore is an upper limit that can be chosen freely (but not too small).

Alternatively SUPERPOSITIONMODE_VELOREDUCTION_ADDITIVEMOTION can be used as *Mode*. In this case the whole correction distance would be up to 367.1°. Since the distance should be as short as possible both modes are equivalent in this case.

2. Slow correction:

Distance = 7.1°

Length = 360° (correction distance)

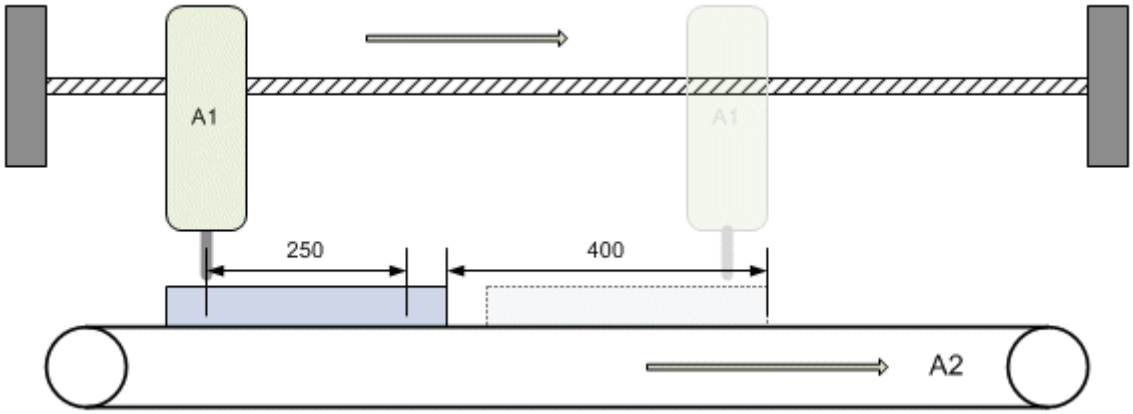
Mode = SUPERPOSITIONMODE_VELOREDUCTION_LIMITEDMOTION

VelocityDiff = 30°/s (velocity reserve)

The *mode* specifies that the correction distance should be utilized fully and the velocity change should be kept as small as possible. The stated value for *VelocityDiff* therefore is an upper limit that can be chosen freely (but not too small).

Drilling unit

A drilling unit should drill two holes in a moving workpiece. Synchronization for the first hole is assumed to be achieved via the flying saw (MC_GearInPos) and is not be considered here. After the first operation the device must be moved by certain distance relative to the moving workpiece.



The drilling unit is to be advanced by 250 mm relative to the workpiece after the first hole has been drilled. Meanwhile the workpiece covers a distance of 400 mm. From this position the drilling unit is once again synchronous with the workpiece and the second hole can be drilled.

Here too two options are available that differ in terms of the velocity change of the drilling device and therefore in the mechanical strain.

Parameterization of function block MC_MoveSuperimposed [▶ 82]:

1. Fast correction:

Distance = 250 mm

Length = 400 mm

Mode = SUPERPOSITIONMODE_LENGTHREDUCTION_ADDITIVEMOTION

VelocityDiff = 500 mm/s (velocity reserve of the drilling device)

The *mode* specifies that the correction distance should be as short as possible. The stated value for *Length* therefore is an upper limit that can be chosen freely (but not too small). The drilling device can travel a larger distance since *Length* refers to the workpiece plus a relative change in position.

2. Slow correction:

Distance = 250 mm

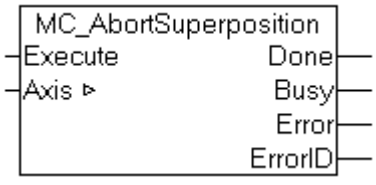
Length = 400 mm

Mode = SUPERPOSITIONMODE_VELOREDUCTION_ADDITIVEMOTION

VelocityDiff = 500 mm/s (velocity reserve of the drilling device)

The *mode* specifies that the correction distance should be utilized fully and the velocity change should be kept as small as possible. The stated value for *VelocityDiff* therefore is an upper limit that can be chosen freely (but not too small). During the change in position the workpiece covers the distance *Length*, the drilling unit travels 650 mm due to the additional correction distance (*Length* + *Distance*).

7.2.3 MC_AbortSuperposition



The *MC_AbortSuperposition* block terminates a superimposed movement started by MC_MoveSuperImposed [▶ 82], without stopping the subordinate axis movement.

A full axis stop can be achieved with MC_Stop [▶ 80] or MC_Halt [▶ 78], if necessary. In this case *MC_AbortSuperposition* does not have to be called.

Inputs

```
VAR_INPUT
Execute : BOOL;
END_VAR
```

Execute	The command is executed with a rising edge and the superimposed movement is completed.

Outputs

```
VAR_OUTPUT
Done : BOOL;
Busy : BOOL;
Error : BOOL;
ErrorID : UDINT;
END_VAR
```

Done	Becomes TRUE when the superimposed movement was successfully terminated.
Busy	Becomes TRUE as soon as the function block is active, and becomes FALSE when it has returned to its initial state.
Error	Becomes TRUE, as soon as an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number.

Inputs/outputs

```
VAR_IN_OUT
Axis : AXIS_REF;
END_VAR
```

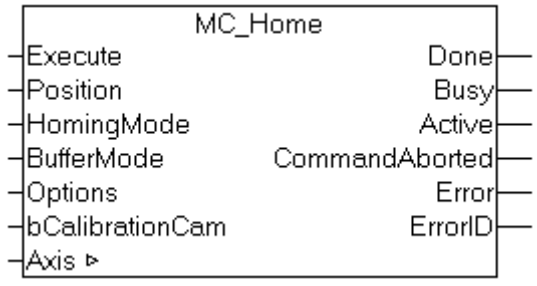
AXIS_REF [▶ 101]

Axis	Axis data structure

The axis data structure of type AXIS_REF [▶ 101] addresses an axis uniquely within the system. Among other parameters it contains the current axis status, including position, velocity or error status.

7.3 Homing

7.3.1 MC_Home



Calibration of the axis (referencing) is carried out with the function block *MC_Home*.

Referencing mode is set in the TwinCAT System Manager via the *Incremental* encoder tab. Depending on the connected encoder system, different procedures are possible (see also Reference mode for inkremental encoder)

Inputs

```
VAR_INPUT
Execute : BOOL;
Position : LREAL := DEFAULT_HOME_POSITION;
HomingMode : MC_HomingMode;
BufferMode : MC_BufferMode;
Options : ST_HomingOptions;
bCalibrationCam : BOOL;
END_VAR
```

MC_BufferMode [▶ 103] MC_HomingMode [▶ 106]

Execute	The command is executed with a rising edge at *Execute* input.	
Position	Absolute reference position to which the axis is set after homing. Alternatively the constant DEFAULT_HOME_POSITION can be used here. In this case, the *Reference position for homing* specified in the TwinCAT System Manager is used. **Notice Since the reference position is generally set during the motion, the axis will not stop exactly at this position. The standstill position deviates by the braking distance of the axis, nevertheless the calibration is exact.**	
HomingMode	HomingMode [▶ 106] determines in which way the calibration is carried out. • MC_DefaultHoming Initiates standard homing. • MC_Direct Sets the axis position directly to *Position* without executing a movement. • MC_ForceCalibration Enforces the "axis is calibrated" state. No movement takes place, and the position remains unchanged. • MC_ResetCalibration Resets the calibration state of the axis. No movement takes place, and the position remains unchanged.	
BufferMode	Currently not implemented - *BufferMode* is analyzed if the axis is already executing another command. The running command can be aborted, or the new command becomes active after the running command. The *BufferMode* also determines the transition condition from the current to the next command.	
Options	The data structure Options includes additional, rarely required parameters. The input can normally remain open.	
Options.	**ClearPositionLag**	ClearPositionLag is only effective in MC_Direct mode. *ClearPositionLag*can optionally be used to set the set and actual positions to the same value. In this case the following error is cleared.
bCalibrationCam	bCalibrationCam reflects the signal of a referencing cam that may enter the controller via a digital input.	

General rules for MC function blocks [▶ 14]

Outputs

```
VAR_OUTPUT
Done : BOOL;
Busy : BOOL;
Active : BOOL;
CommandAborted : BOOL;
Error : BOOL;
ErrorID : UDINT;
END_VAR
```

Done	The *Done* output becomes TRUE, if the axis was calibrated and has come to a standstill.
Busy	The *Busy* output becomes TRUE when the command is started with *Execute* and remains TRUE as long as the movement command is processed. If *Busy* becomes FALSE again, the function block is ready for a new job. At the same time one of the outputs, *Done*, *CommandAborted* or *Error*, is set.
Active	Currently not implemented - Active indicates that the command is running. If the command was queued, it becomes active once a running command is completed.
CommandAborted	Becomes TRUE, if the command could not be fully executed.
Error	Becomes TRUE if an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number.

General rules for MC function blocks [▶ 14]

Inputs/outputs

```
VAR_IN_OUT
Axis : AXIS_REF;
END_VAR
```

AXIS_REF [▶ 101]

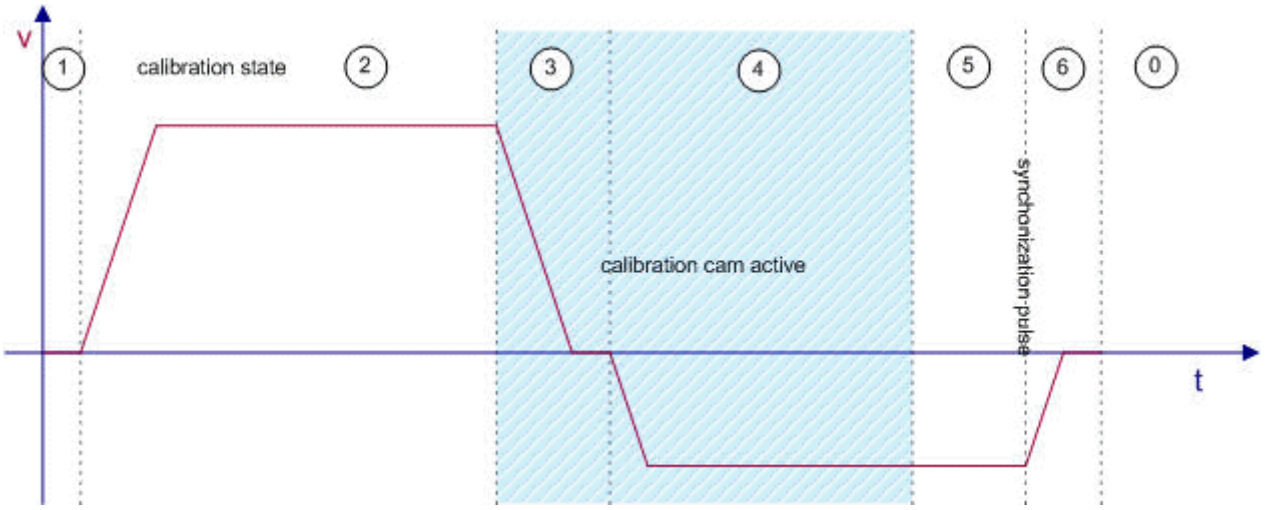
Axis	Axis data structure

The axis data structure of type AXIS_REF [▶ 101] addresses an axis uniquely within the system. Among other parameters it contains the current axis status, including position, velocity or error status.

Note

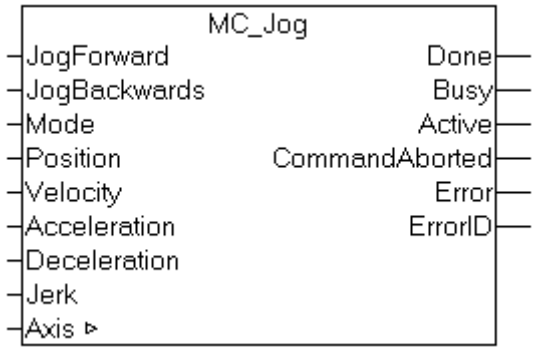
The referencing process has several phases. The referencing state (calibration state) is signaled in the cyclic interface of the axis (Axis.NcToPlc.HomingState). The following diagram illustrates the individual process phases after starting of the *MC_Home* block.

If an axis is to be referenced without reference cam, i.e. only based on the sync pulse of the sensor, the reference cam can be simulated via the PLC program. The bCalibrationCam signal is initially activated and then cancelled, if Axis.NcToPlc.HomingState [▶ 102] is equal or greater 4.



7.4 Manual motion

7.4.1 MC_Jog



The *MC_Jog* function block enables an axis to be moved via manual keys. The key signal can be linked directly with the *JogForward* and *JogBackwards* inputs. The required operating mode is specified via the mode input. An inching mode for moving the axis by a specified distance whenever the key is pressed is also available. The velocity and dynamics of the motion can be specified depending on the mode.

Inputs

```
VAR_INPUT
JogForward : BOOL;
JogBackwards : BOOL;
Mode : E_JogMode;
Position : LREAL;
Velocity : LREAL;
Acceleration : LREAL;
Deceleration : LREAL;
Jerk : LREAL;
END_VAR
```

E_JogMode [▶ 108]

JogForward	The command is executed with rising edge, and the axis is moved in positive direction of travel. Depending on the operation mode (see mode), the axis moves as long as the signal remains TRUE, or it stops automatically after a specified distance. During the motion no further signal edges are

	accepted (this includes the JogBackwards input). If signal edges occur simultaneously at the JogForward and JogBackwards inputs, JogForward has priority.
JogBackwards	The command is executed with rising edge and the axis moved in negative direction of travel. JogForward and JogBackwards should be triggered alternatively, although they are also mutually locked internally.
Mode	The Mode [▶ 108] input specifies the mode for manual operation. *MC_JOGMODE_STANDARD_SLOW:* the axis moves as long as the signal at one of the jog inputs is TRUE. The *low velocity for manual functions* specified in the TwinCAT System Manager and standard dynamics are used. In this operation mode the position, velocity and dynamics data specified in the function block have no effect. *MC_JOGMODE_STANDARD_FAST:* the axis moves as long as the signal at one of the jog inputs is TRUE. The *high velocity for manual functions* specified in the TwinCAT System Manager and standard dynamics are used. In this mode the position, velocity and dynamics data specified in the function block have no effect. *MC_JOGMODE_CONTINOUS:* the axis is moved as long as the signal at one of the jog inputs is TRUE. The velocity and dynamics data specified by the user are used. The position has no effect. *MC_JOGMODE_INCHING:* with rising edge at one of the jog inputs the axis is moved by a certain distance, which is specified via the position input. The axis stops automatically, irrespective of the state of the jog inputs. A new movement step is only executed once a further rising edge is encountered. With each start the velocity and dynamics data specified by the user are used. *MC_JOGMODE_INCHING_MODULO:* with rising edge at one of the jog inputs the axis is moved by a certain distance which is specified via the position input. The axis position will snap to an integer multiple of the position parameter. The axis stops automatically, irrespective of the state of the jog inputs. A new movement step is only executed once a further rising edge is encountered. With each start the velocity and dynamics data specified by the user are used.
Position	relative distance for movements in *MC_JOGMODE_INCHING* operation mode.
Velocity	Maximum travel velocity (>0).
Acceleration	Acceleration (≥0). If the value is 0, the standard acceleration from the axis configuration in the System Manager is used.
Deceleration	Deceleration (≥0). If the value is 0, the standard deceleration from the axis configuration in the System Manager is used.
Jerk	Jerk (≥0). If the value is 0, the standard jerk from the axis configuration in the System Manager is used.

The parameters *Position*, *Velocity*, *Acceleration*, *Deceleration* and *Jerk* are not used in the operation modes *MC_JOGMODE_STANDARD_SLOW* and *MC_JOGMODE_STANDARD_FAST* and can remain unassigned.

Outputs

```
VAR_OUTPUT
Done : BOOL;
Busy : BOOL;
CommandAborted : BOOL;
Error : BOOL;
ErrorID : UDINT;
END_VAR
```

Done	Becomes TRUE if a movement is completed successfully.
Busy	Becomes TRUE as soon as the function block is active, and becomes FALSE when it has returned to its initial state. Only then can a further edge be accepted at the jog inputs.
Active	Active indicates that the axis is moved via the jog function.
CommandAborted	Becomes TRUE if the process is interrupted by an external event, e.g. by the call up of MC_Stop [▶ 80].
Error	Becomes TRUE if an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number.

Inputs/outputs

```
VAR_IN_OUT
Axis : AXIS_REF;
END_VAR
```

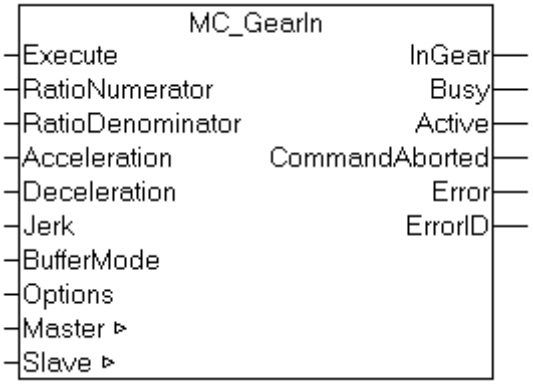
AXIS_REF [▶ 101]

Axis	Axis data structure

The axis data structure of type AXIS_REF [▶ 101] addresses an axis unambiguously within the system. Among other parameters it contains the current axis status, including position, velocity or error state.

7.5 Axis coupling

7.5.1 MC_GearIn



The function block *MC_GearIn* activates a linear master-slave coupling (gear coupling). The block accepts a fixed gear ratio in numerator/denominator format.

The slave axis can be coupled to the master axis when stationary. This block cannot be used for synchronization while the master axis is in motion. In this case the *Flying Saw* block MC_GearInVelo or MC_GearInPos can be used.

The slave axis can be uncoupled with the function block MC_GearOut [▶ 97]. If the slave is decoupled while it is moving, then it retains its velocity and can be halted using MC_Stop [▶ 80]or MC_Halt [▶ 78].

Alternatively, the block MC_GearInDyn [▶ 95] with dynamically variable gear ratio is available.

Inputs

```
VAR_INPUT
Execute : BOOL;
RatioNumerator : LREAL;
RatioDenominator : UINT;
Acceleration : LREAL;
Deceleration : LREAL;
Jerk : LREAL;
BufferMode : MC_BufferMode;
Options : ST_GearInOptions;
END_VAR
```

MC_BufferMode [▶ 103]

Execute	The command is executed with a rising edge at input *Execute*.
RatioNumerator	Gear ratio numerator. Alternatively, the gear ratio can be specified in the numerator as a floating point value, if the denominator is 1.
RatioDenominator	Gear ratio denominator
Acceleration	Acceleration (≥0). (currently not implemented)
Deceleration	Deceleration (≥0). (currently not implemented)
Jerk	Jerk (≥0). (currently not implemented)
BufferMode	Currently not implemented
Options	Currently not implemented

For a 1:4 ratio the *RatioNumerator* must be 1, the *RatioDenominator* must be 4. Alternatively, the *RatioDenominator* may be 1, and the gear ratio can be specified as floating-point number 0.25 under *RatioNumerator*. The *RatioNumerator* may be negative.

Outputs

```
VAR_OUTPUT
InGear : BOOL;
Busy : BOOL;
Active : BOOL;
CommandAborted : BOOL;
Error : BOOL;
ErrorID : UDINT;
END_VAR
```

InGear	Becomes TRUE, if the coupling was successful.
Busy	The *Busy* output becomes TRUE when the command is started with *Execute* and remains TRUE as long as the command is processed. If *Busy* becomes FALSE again, the function block is ready for a new job. At the same time one of the outputs *InGear*, *CommandAborted* or *Error* is set.

Active	Active indicates that the command is executed (currently Active=Busy, see BufferMode)
CommandAborted	Becomes TRUE, if the command could not be fully executed. The axis may have become decoupled during the coupling process (simultaneous command execution).
Error	Becomes TRUE if an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number.

Inputs/outputs

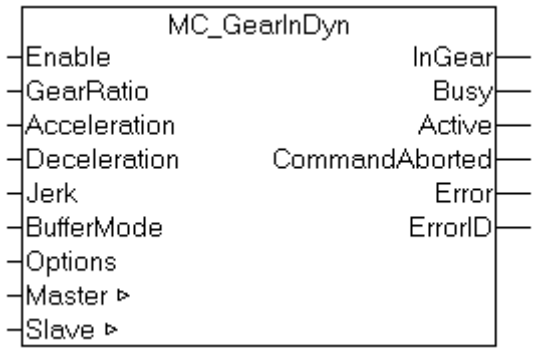
```
VAR_IN_OUT
Master : AXIS_REF;
Slave : AXIS_REF;
END_VAR
```

AXIS_REF [▶ 101] AXIS_REF [▶ 101]

Master	Master axis data structure.
Slave	Axis data structure of the Slave.

The axis data structure of type AXIS_REF [▶ 101] addresses an axis uniquely within the system. Among other parameters it contains the current axis status, including position, velocity or error status.

7.5.2 MC_GearInDyn



The function block *MC_GearIn* activates a linear master-slave coupling (gear coupling). The gear ratio can be adjusted dynamically during each PLC cycle. Hence a controlled master/slave coupling can be build up. The *Acceleration* parameter has a limiting effect in situations with large gear ratio variations.

The slave axis can be uncoupled with the function block MC_GearOut [▶ 97]. If the slave is decoupled while it is moving, then it retains its velocity and can be halted using MC_Stop [▶ 80]or MC_Halt [▶ 78].

Alternatively, the block MC_GearIn [▶ 93] with dynamically variable gear ratio is available.

Inputs

```
VAR_INPUT
Enable : BOOL;
GearRatio : LREAL;
Acceleration : LREAL;
Deceleration : LREAL;
Jerk : LREAL;
BufferMode : MC_BufferMode;
Options : ST_GearInDynOptions;
END_VAR
```

MC_BufferMode [▶ 103]

Enable	Coupling is activated with a rising edge at input *Enable*. The gear ratio can be changed cyclically as long as *Enable* is TRUE. The command is terminated if *Enable* becomes FALSE after coupling. The gear ratio is frozen at its last value, but the slave is not decoupled.
GearRatio	Gear ratio as floating point value. The gear ratio can be changed cyclically as long as *Enable* is TRUE. If *ENABLE* is FALSE, the gear ratio remains unchanged.
Acceleration	Acceleration (≥0). If the value is 0, the standard acceleration from the axis configuration in the System Manager is used. The parameter limits the acceleration of the slave in situations with large gear ratio variations. The maximum acceleration is only reached at the maximum master velocity. Otherwise the slave acceleration is below this value when the gear ratio changes significantly.
Deceleration	Deceleration (≥0). (Not implemented)
Jerk	Jerk (≥0). (Not implemented)
BufferMode	Currently not implemented
Options	Currently not implemented

Outputs

```
VAR_OUTPUT
InGear : BOOL;
Busy : BOOL;
Active : BOOL;
CommandAborted : BOOL;
Error : BOOL;
ErrorID : UDINT;
END_VAR
```

InGear	Becomes TRUE, if the coupling was successful.
Busy	The *Busy* output becomes TRUE when the command is started with *Enable* and remains TRUE as long as the command is processed. If *Busy* becomes FALSE again, the function block is ready for a new job. At the same time one of the outputs *InGear, CommandAborted* or *Error* is set.
Active	Active indicates that the command is executed (currently Active=Busy, see BufferMode)
CommandAborted	Becomes TRUE, if the command could not be fully executed. The axis may have become decoupled during the coupling process (simultaneous command execution).
Error	Becomes TRUE if an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number.

Inputs/outputs

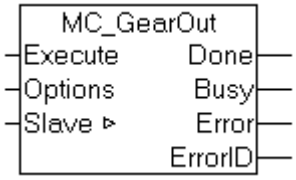
```
VAR_IN_OUT
Master : AXIS_REF;
Slave : AXIS_REF;
END_VAR
```

AXIS_REF [▶ 101] AXIS_REF [▶ 101]

Master	Master axis data structure.
Slave	Axis data structure of the Slave.

The axis data structure of type AXIS_REF [▶ 101] addresses an axis uniquely within the system. Among other parameters it contains the current axis status, including position, velocity or error status.

7.5.3 MC_GearOut



The function block *MC_GearOut* deactivates a master-slave coupling.

⚠ WARNING

No standstill of the axis due to decoupling

When a slave axis is uncoupled during the movement, it is not stopped automatically but reaches a constant velocity at which it continues to travel infinitely.

You can stop the axis with the function blocks MC_Halt [▶ 78] or MC_Stop [▶ 80].

NOTICE

If the setpoint generator type of the axis is set to *"7 phases (optimized)"*, the slave axis assumes an acceleration-free state after uncoupling and continues to move with the resulting constant velocity. There is no positioning based on the master travel path calculated with the coupling factor. Instead, the behavior matches the behavior after a MC_MoveVelocity command. In TwinCAT 2.10, the setpoint generator type can be selected by the user. From TwinCAT 2.11, the setpoint generator type is set to *"7 phases (optimized)"*. The behavior described here is the result of a project update from TwinCAT 2.10 to TwinCAT 2.11. Depending on the circumstances, an update of existing applications to version 2.11 may necessitate an adaptation of the PLC program.

Inputs

```
VAR_INPUT
    Execute  : BOOL;
    Options  : ST_GearOutOptions;
END_VAR
```

Execute	The command is executed with a rising edge at input *Execute*.
Options	Currently not implemented

Outputs

```
VAR_OUTPUT
    Done     : BOOL;
    Busy     : BOOL;
    Error    : BOOL;
    ErrorID  : UDINT;
END_VAR
```

Done	Becomes TRUE, if the axis was successfully uncoupled.
Busy	The *Busy* output becomes TRUE when the command is started with *Execute* and remains TRUE as long as the command is processed. If *Busy* becomes FALSE again, the function block is ready for a new job. At the same time one of the outputs, *Done* or *Error*, is set.
Error	Becomes TRUE if an error occurs.

ErrorID	If the error output is set, this parameter supplies the error number.

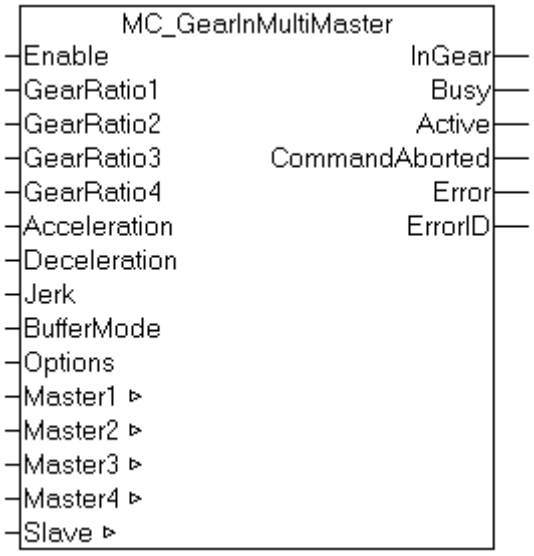
Inputs/outputs

```
VAR_IN_OUT
    Slave : AXIS_REF;
END_VAR
```

Slave	Axis data structure of the Slave [▶ 101].

The axis data structure of type AXIS_REF [▶ 101] addresses an axis uniquely within the system. Among other parameters it contains the current axis status, including position, velocity or error status.

7.5.4 MC_GearInMultiMaster



The function block *MC_GearInMultiMaster* is used to activate linear master/slave coupling (gear coupling) for up to four different master axes. The gear ratio can be adjusted dynamically during each PLC cycle. The slave movement is determined by the superimposed master movements. The *Acceleration* parameter has a limiting effect in situations with large gear ratio variations.

The slave axis can be uncoupled with the function block MC_GearOut [▶ 97]. If the slave is decoupled while it is moving, then it retains its velocity and can be halted using MC_Stop [▶ 80].

If fewer than four masters are used, an empty data structure can be transferred for parameters *Master2* to *Master4* (the axis ID must be 0).

Inputs

```
VAR_INPUT
Enable : BOOL;
GearRatio1 : LREAL;
GearRatio2 : LREAL;
GearRatio3 : LREAL;
GearRatio4 : LREAL;
Acceleration : LREAL;
Deceleration : LREAL;
Jerk : LREAL;
BufferMode : MC_BufferMode;
Options : ST_GearInMultiMasterOptions;
END_VAR
```

Enable	Coupling is activated with a rising edge at input *Enable*. The gear ratio can be changed cyclically as long as *Enable* is TRUE.

		The command is terminated if *Enable* becomes FALSE after coupling. The gear ratio is frozen at its last value, but the slave is not decoupled.
GearRatio1		Gear ratio as floating point value for the first master axis. The gear ratio can be changed cyclically as long as *Enable* is TRUE. If *ENABLE* is FALSE, the gear ratio remains unchanged.
GearRatio2		Gear ratio as floating point value for the second master axis. The gear ratio can be changed cyclically as long as *Enable* is TRUE. If *ENABLE* is FALSE, the gear ratio remains unchanged.
GearRatio3		Gear ratio as floating point value for the third master axis. The gear ratio can be changed cyclically as long as *Enable* is TRUE. If *ENABLE* is FALSE, the gear ratio remains unchanged.
GearRatio4		Gear ratio as floating point value for the fourth master axis. The gear ratio can be changed cyclically as long as *Enable* is TRUE. If *ENABLE* is FALSE, the gear ratio remains unchanged.
Acceleration		Acceleration (≥0). If the value is 0, the standard acceleration from the axis configuration in the System Manager is used. The parameter limits the acceleration of the slave in situations with large gear ratio variations.
Deceleration		Deceleration (≥0). If the value is 0, the standard deceleration from the axis configuration in the System Manager is used. The parameter limits the deceleration of the slave in situations with large gear ratio variations. **Used only for the option "Advanced Slave Dynamics".**
Jerk		Jerk (≥0). If the value is 0, the standard jerk from the axis configuration in the System Manager is used. The parameter limits the jerk of the slave in situations with large gear ratio variations. **Used only for the option "Advanced Slave Dynamics".**
BufferMode		Currently not implemented
Options.	**AdvancedSlaveDynamics**	Exchanges the internal algorithm of the function block. This makes it possible to synchronise to masters already in motion. Acceleration and deceleration should only be parameterized symmetrically. If jerk presets are too large, this is reduced to the extent that a change from zero to the parameterized acceleration / deceleration can take place in one NC cycle. The resolution of the acceleration / deceleration thus depends directly on the suitable parameterization of the jerk value.

Outputs

```
VAR_OUTPUT
InGear : BOOL;
Busy : BOOL;
Active : BOOL;
CommandAborted : BOOL;
Error : BOOL;
ErrorID : UDINT;
END_VAR
```

InGear	Becomes TRUE, if the coupling was successful.

Busy	The *Busy* output becomes TRUE when the command is started with *Enable* and remains TRUE as long as the command is processed. If *Busy* becomes FALSE again, the function block is ready for a new job. At the same time one of the outputs *InGear, CommandAborted* or *Error* is set.
Active	Active indicates that the command is executed (currently Active=Busy, see BufferMode)
CommandAborted	Becomes TRUE, if the command could not be fully executed. The axis may have become decoupled during the coupling process (simultaneous command execution).
Error	Becomes TRUE if an error occurs.
ErrorID	If the error output is set, this parameter supplies the error number.

Inputs/outputs

```
VAR_IN_OUT
Master1 : AXIS_REF;
Master2 : AXIS_REF;
Master3 : AXIS_REF;
Master4 : AXIS_REF;
Slave : AXIS_REF;
END_VAR
```

AXIS_REF [▶ 101] AXIS_REF [▶ 101] AXIS_REF [▶ 101] AXIS_REF [▶ 101] AXIS_REF [▶ 101]

Master1	Axis data structure of the first master.
Master2	Axis data structure of the second master.
Master3	Axis data structure of the third master.
Master4	Axis data structure of the fourth master.
Slave	Axis data structure of the Slave.

The axis data structure of type AXIS_REF [▶ 101] addresses an axis uniquely within the system. Among other parameters it contains the current axis status, including position, velocity or error status.

8 Data types

8.1 Axis interface

8.1.1 Data type AXIS_REF

The AXIS_REF data type contains axis information. AXIS_REF is an interface between the PLC and the NC. It is added to MC function blocks as axis reference.

```
TYPE AXIS_REF :
VAR_INPUT
PlcToNc    AT %Q* : PLCTONC_AXIS_REF;
END_VAR
VAR_OUTPUT
NcToPlc    AT %I* : NCTOPLC_AXIS_REF;
ADS               : ST_AdsAddress;
Status            : ST_AxisStatus;
END_VAR
END_TYPE
```

ST_AxisStatus [▶ 109] NCTOPLC_AXIS_REF [▶ 102] PLCTONC_AXIS_REF [▶ 102]

AXIS_REF elements

PlcToNc: PlcToNc [▶ 102] is a data structure that is cyclically exchanged between PLC and NC. Via this data structure the MC function blocks communicate with the NC and send control information from the PLC to the NC. This data structure is automatically placed in the output process image of the PLC and must be linked in TwinCAT System Manager with the input process image of an NC axis.

NcToPlc: NcToPlc [▶ 102] is a data structure that is cyclically exchanged between PLC and NC. Via this data structure the MC function blocks communicate with the NC and receive status information from the NC. This data structure is automatically placed in the input process image of the PLC and must be linked in TwinCAT System Manager with the output process image of an NC axis.

The NcToPlc [▶ 102] structure contains all main state data for an axis such as position, velocity and instruction state. Since data exchange takes place cyclically, the PLC can access the current axis state at any time without additional communication effort.

ADS: The ADS data structure contains the ADS communication parameters for an axis that are required for direct ADS communication. Normally this structure does not have to be populated. The user can use it to stored information for controlling an axis on another target system or via a special port number.

Status: The Status data structure [▶ 109] contains additional or processed status information for an axis. This data structure is not refreshed cyclically, but has to be updated through the PLC program. The easiest way to achieve this is by calling MC_ReadStatus [▶ 30] or, alternatively, by calling the action *ReadStatus* of AXIS_REF:

Example:

```
VAR
Axis1 : AXIS_REF (* axis data structure for Axis-1 *)
END_VAR

(* program code at the beginning of each PLC cycle *)
Axis1.ReadStatus;
```

ReadStatus should be called once at the start of each PLC cycle. The current status information can then be accessed in AXIS_REF from the whole PLC program. Within a cycle the status does not change.

8.1.2 Data type NCTOPLC_AXIS_REF

The data structure NCTOPLC_AXIS_REF is part of the AXIS_REF [▶ 101] data structure and is automatically updated by the NC, so that updated information is available during each PLC cycle. NCTOPLC_AXIS_REF is also referred to as axis interface between NC and PLC.

```
TYPE NCTOPLC_AXIS_REF
STRUCT
StateDWord            : DWORD; (* Status double word *)
ErrorCode             : DWORD; (* Axis error code *)
AxisState             : DWORD; (* Axis moving status *)
AxisModeConfirmation  : DWORD; (* Axis mode confirmation (feedback from NC) *)
HomingState           : DWORD; (* State of axis calibration (homing) *)
CoupleState           : DWORD; (* Axis coupling state *)
SvbEntries            : DWORD; (* SVB entries/orders (SVB = Set preparation task) *)
SafEntries            : DWORD; (* SAF entries/orders (SAF = Set execution task) *)
AxisId                : DWORD; (* Axis ID *)
OpModeDWord           : DWORD; (* Current operation mode *)
ActiveControlLoopIndex: WORD;  (* Active control loop index *)
ControlLoopIndex      : WORD;  (* Axis control loop index (0, 1, 2, when multiple control loops are
used) *)
ActPos                : LREAL; (* Actual position (absolut value from NC) *)
ModuloActPos          : LREAL; (* Actual modulo position *)
ModuloActTurns        : DINT;  (* Actual modulo turns *)
ActVelo               : LREAL; (* Actual velocity *)
PosDiff               : LREAL; (* Position difference (lag distance) *)
SetPos                : LREAL; (* Setpoint position *)
SetVelo               : LREAL; (* Setpoint velocity *)
SetAcc                : LREAL; (* Setpoint acceleration *)
TargetPos             : LREAL; (* Estimated target position *)
ModuloSetPos          : LREAL; (* Setpoint modulo position *)
ModuloSetTurns        : DINT;  (* Setpoint modulo turns *)
CmdNo                 : WORD;  (* Continuous actual command number *)
CmdState              : WORD;  (* Command state *)
END_STRUCT
END_TYPE
```

Extended description of the TYPE NCTOPLC_AXLESTRUCT2 data structure

8.1.3 Data type PLCTONC_AXIS_REF

The data structure PLCTONC_AXIS_REF is part of the AXIS_REF [▶ 101] data structure and cyclically transfers information to the NC. PLCTONC_AXIS_REF is also referred to as axis interface between PLC and NC.

```
TYPE PLCTONC_AXIS_REF
STRUCT
ControlDWord       : DWORD; (* Control double word *)
Override           : DWORD; (* Velocity override *)
AxisModeRequest    : DWORD; (* Axis operating mode (PLC request) *)
AxisModeDWord      : DWORD; (* optional mode parameter *)
AxisModeLReal      : LREAL; (* optional mode parameter *)
PositionCorrection : LREAL; (* Correction value for current position *)
ExtSetPos          : LREAL; (* external position setpoint *)
ExtSetVelo         : LREAL; (* external velocity setpoint *)
ExtSetAcc          : LREAL; (* external acceleration setpoint *)
ExtSetDirection    : DINT; (* external direction setpoint *)
Reserved1          : DWORD; (* reserved *)
ExtControllerOutput: LREAL; (* external controller output *)
GearRatio1         : LREAL; (* Gear ratio for dynamic multi master coupling modes *)
GearRatio2         : LREAL; (* Gear ratio for dynamic multi master coupling modes *)
GearRatio3         : LREAL; (* Gear ratio for dynamic multi master coupling modes *)
GearRatio4         : LREAL; (* Gear ratio for dynamic multi master coupling modes *)
MapState           : BYTE; (* reserved - internal use *)
Reserved_HIDDEN    : ARRAY [105..127] OF BYTE;
END_STRUCT
END_TYPE
```

Extended description of the TYPE PLCTONC_AXLESTRUCT data structure

8.2 Motion function blocks

8.2.1 Data type MC_BufferMode

The data type *MC_BufferMode* is used with various function blocks from the motion control library. BufferMode is used to specify how successive travel commands are to be processed.

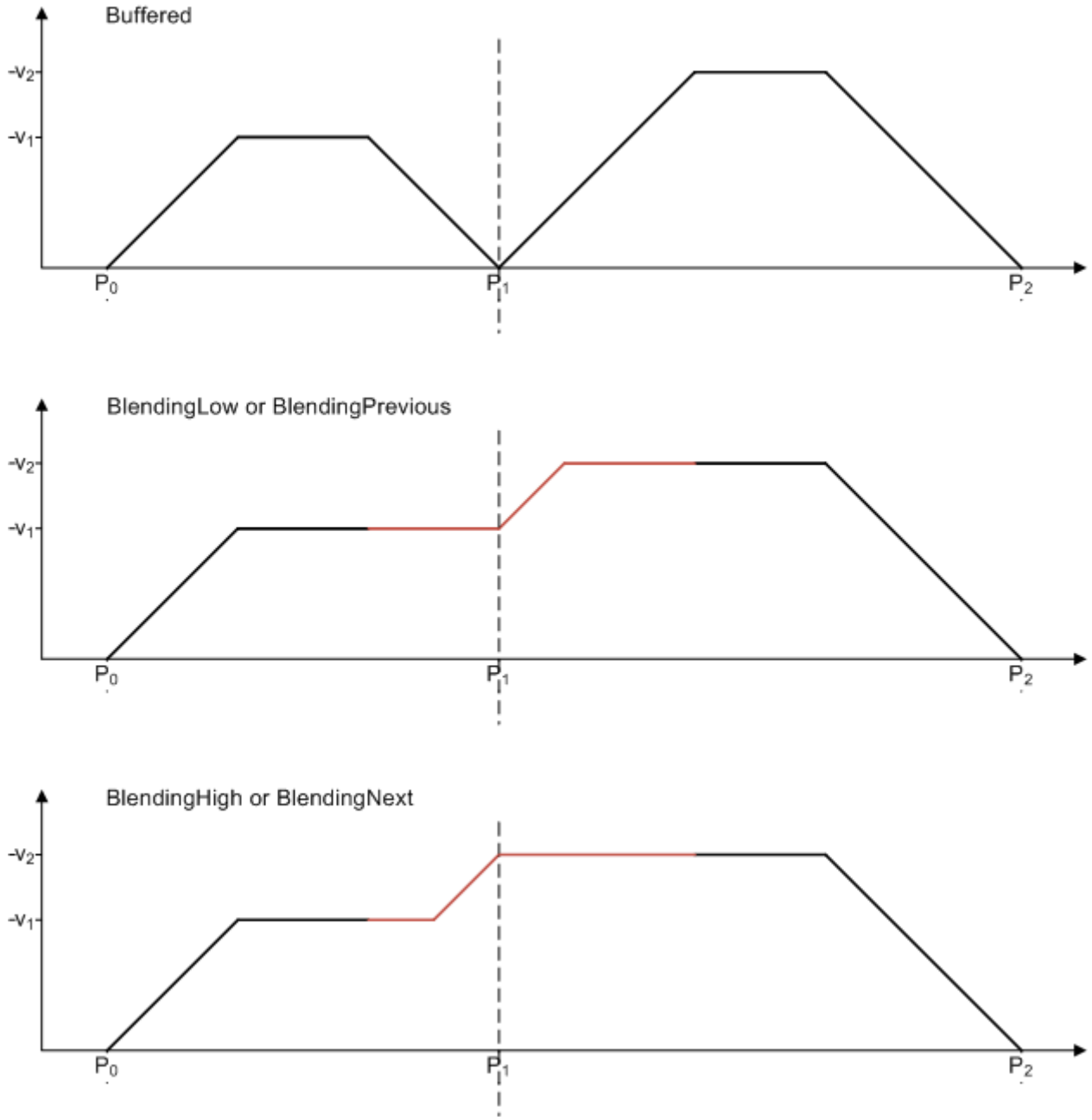
```
TYPE MC_BufferMode :
(
MC_Aborting,
MC_Buffered,
MC_BlendingLow,
MC_BlendingPrevious,
MC_BlendingNext,
MC_BlendingHigh
);
END_TYPE
```

see also: BufferMode in the section on general rules for MC function blocks [▶ 14]

A second function block is required to use the buffer mode. It is not possible to trigger a move block with new parameters while it is active.

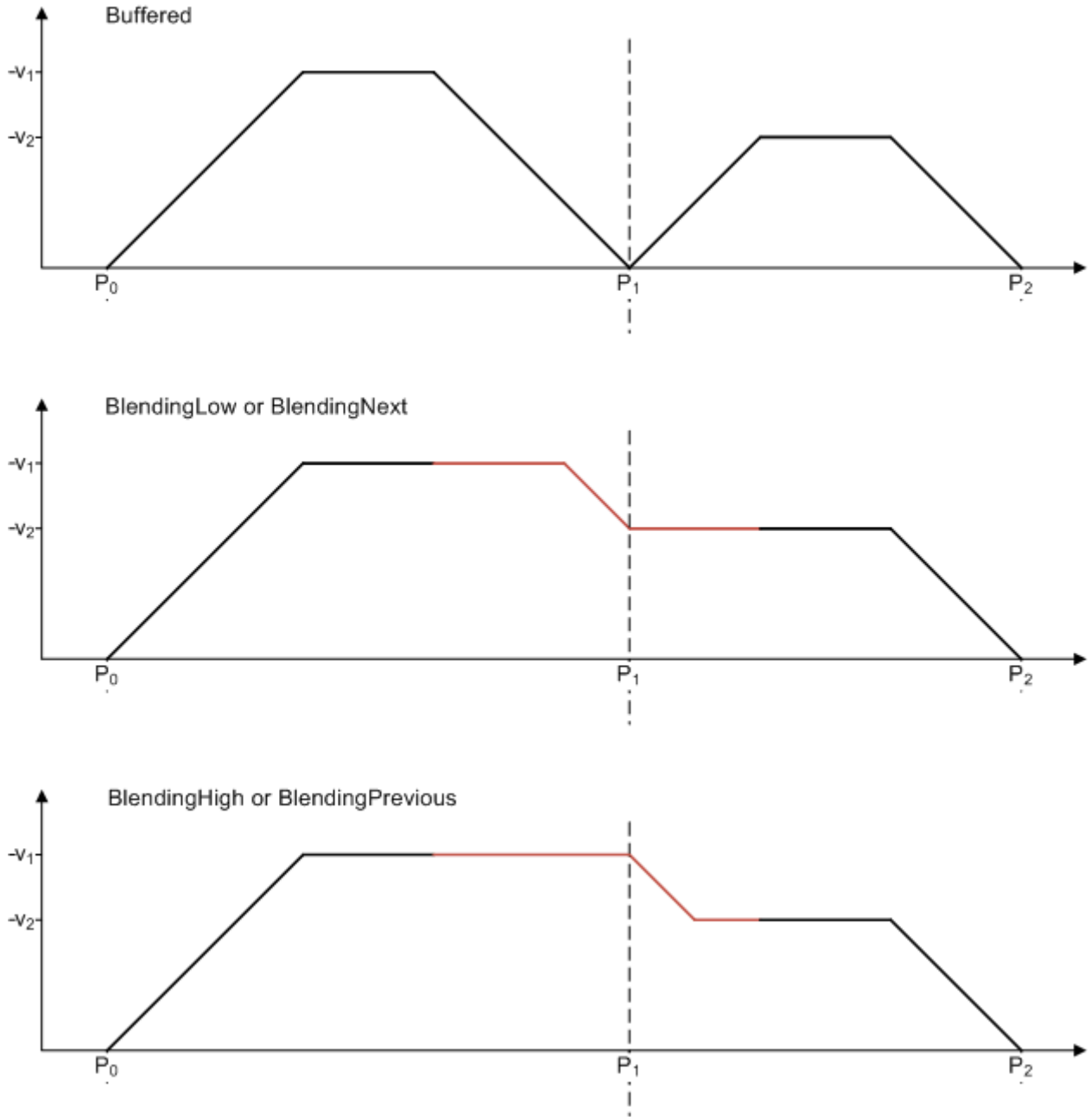
Examples:

In the following example, a move command is used to move an axis from position P_0 to P_1 and then to P_2. The second command is issued during the movement to P_1, but before the braking ramp with different buffer modes. The reference point for the different velocity profiles is always P_1. The mode specifies the velocity v_1 or v_2 at this point.



Since the speed of the first command is lower than the second, the modes BlendingLow/BlendingPrevious and BlendingHigh/BlendingNext have the same result.

The difference in the next example is that the speed of the second command is lower than the first. Now, the modes BlendingLow/BlendingNext and BlendingHigh/BlendingPrevious are equivalent.



The velocity profiles described here assume that the following command is issued in time, i.e. before the braking ramp of the first command. Otherwise, blending is implemented as best as possible.

8.2.2 Data type MC_Direction

```
TYPE MC_Direction :
(
MC_Positive_Direction := 1,
MC_Shortest_Way ,
MC_Negative_Direction,
MC_Current_Direction
);
END_TYPE
```

This listing type contains the possible directions of travel for the function blocks MC_MoveVelocity [▶ 72] and MC_MoveModulo [▶ 64].

8.2.3 Data type MC_HomingMode

The data type *MC_HomingMode* is used for parameterizing the function block MC_Home [▶ 88]

```
TYPE MC_HomingMode :
(
MC_DefaultHoming,     (* default homing as defined in the SystemManager encoder parameters *)
MC_AbsSwitch,         (* not implemented - Absolute Switch homing plus Limit switches *)
MC_LimitSwitch,       (* not implemented - Homing against Limit switches *)
MC_RefPulse,          (* not implemented - Homing using encoder Reference Pulse "Zero Mark" *)
MC_Direct,            (* Static Homing forcing position from user reference *)
MC_Absolute,          (* not implemented - Static Homing forcing position from absolute encoder *)
MC_Block,             (* not implemented - Homing against hardware parts blocking movement *)
MC_ForceCalibration, (* set the calibration flag without perfomring any motion or changing the
position *)
MC_ResetCalibration  (* resets the calibration flag without perfomring any motion or changing the
position *)
);
END_TYPE
```

8.2.4 Data type E_SuperpositionMode

```
TYPE E_SuperpositionMode :
(
SUPERPOSITIONMODE_VELOREDUCTION_ADDITIVEMOTION := 1,
SUPERPOSITIONMODE_VELOREDUCTION_LIMITEDMOTION,
SUPERPOSITIONMODE_LENGTHREDUCTION_ADDITIVEMOTION,
SUPERPOSITIONMODE_LENGTHREDUCTION_LIMITEDMOTION,
SUPERPOSITIONMODE_ACCREDUCTION_ADDITIVEMOTION,   (* from TwinCAT 2.11 *)
SUPERPOSITIONMODE_ACCREDUCTION_LIMITEDMOTION     (* from TwinCAT 2.11 *)
);
END_TYPE
```

E_SuperpositionMode determines how a superimposed motion is carried out with the function block MC_MoveSuperImposed [▶ 82].

The modes referred to as *Veloreduction* execute a superimposed movement with minimum velocity change, preferentially over the full parameterized compensation section. Conversely, the modes referred to as *Lengthreduction* use the maximum possible velocity and therefore reduce the required distance. In both cases same distance is compensated.

In cases referred to as *Additivemotion*, the superimposed axis executes a longer or shorter movement than indicated by *Length*, with the difference described by *Distance*. These modes are used, for example, if the *Length* parameter refers to a reference axis and the superimposed axis may move by a longer or shorter distance in comparison.

In cases referred to as *Limitedmotion*, the superposition is completed within the parameterized distance. These modes are used, for example, if the *Length* parameter refers to the superimposed axis itself. With these modes it should be noted that the superimposed *Distance* must be significantly shorter than the available *Length*.

SUPERPOSITIONMODE_VELOREDUCTION_ADDITIVEMOTION:

The superimposed motion takes place over the whole *Length*. The specified maximum change in velocity VelocityDiff is reduced in order to reach the required *Distance* over this length.

The *Length* is based on a reference axis without superimposed motion (e.g. master axis). The travel path of the axis affected by this compensation is *Length* + *Distance*.

SUPERPOSITIONMODE_VELOREDUCTION_LIMITEDMOTION:

The superimposed motion takes place over the whole Length. The specified maximum change in velocity VelocityDiff is reduced in order to reach the required *Distance* over this length.

The *Length* refers to the axis affected by the compensation. During compensation, the travel path of this axis is *Length*.

SUPERPOSITIONMODE_LENGTHREDUCTION_ADDITIVEMOTION:

The distance of the superimposed motion is as short as possible and the speed is as high as possible. Although neither the maximum velocity change *VelocityDiff* or the maximum *Length* are exceeded.

The *Length* is based on a reference axis without superimposed motion (e.g. master axis). The maximum travel path of the axis affected by this compensation is *Length* + *Distance*.

SUPERPOSITIONMODE_LENGTHREDUCTION_LIMITEDMOTION:

The distance of the superimposed motion is as short as possible and the speed is as high as possible. Although neither the maximum velocity change *VelocityDiff* or the maximum *Length* are exceeded.

The *Length* refers to the axis affected by the compensation. During compensation, the maximum travel path of this axis is *Length*.

SUPERPOSITIONMODE_ACCREDUCTION_ADDITIVEMOTION (from TwinCAT 2.11)

The superimposed motion takes place over the whole *Length*. The specified maximum acceleration (parameter *Acceleration* or *Deceleration)* is reduced as far as possible, in order to reach the specified *Distance* on this path.

The *Length* is based on a reference axis without superimposed motion (e.g. master axis). The travel path of the axis affected by this compensation is *Length* + *Distance*.

SUPERPOSITIONMODE_ACCREDUCTION_LIMITEDMOTION (from TwinCAT 2.11)

The superimposed motion takes place over the whole Length. The specified maximum acceleration (parameter *Acceleration* or *Deceleration)* is reduced as far as possible, in order to reach the specified *Distance* on this path.

The *Length* refers to the axis affected by the compensation. During compensation, the travel path of this axis is *Length*.

8.2.5 Data typeST_SuperpositionOptions

```
TYPE ST_SuperpositionOptions :
STRUCT
AbortOption : E_SuperpositionAbortOption;
END_STRUCT
END_TYPE

TYPE E_SuperpositionAbortOption :
(
SUPERPOSITIONOPTION_ABORTATSTANDSTILL := 0,
SUPERPOSITIONOPTION_RESUMEAFTERSTANDSTILL,
SUPERPOSITIONOPTION_RESUMEAFTERMOTIONSTOP
);
END_TYPE
```

AbortOption

AbortOption is an optional parameter of the MC_MoveSuperimposed [▶ 82] block that specifies the behavior of a superimposed movement in the case of a standstill of the main movement.

SUPERPOSITIONOPTION_ABORTATSTANDSTILL:

The superimposed movement is aborted as soon as the subordinate movement leads to a standstill of the axis. The only exception to this is a standstill caused by a speed override of zero. In this case the superimposed movement is also continued as soon as the override is not equal to zero. *AbortAtStandstill* is the standard behavior if the option is not assigned by the user.

SUPERPOSITIONOPTION_RESUMEAFTERSTANDSTILL:

The superimposed movement is not aborted in the case of a temporary standstill of the main movement, but is continued as soon as the axis moves again. This can occur in particular in the case of a reversal of direction or with cam disc movements. The superimposed movement is terminated only if the target position of the axis has been reached or the axis has been stopped.

SUPERPOSITIONOPTION_RESUMEAFTERMOTIONSTOP:

The superimposed movement is not aborted in the case of a standstill of the main movement, even if the axis has reached its target position or has been stopped. In this case, the superimposed movement is continued after the axis restarts.

This case is not of importance if the superimposed movement is applied to a slave axis, since this cannot be started or stopped actively. In the case of slave axes, the modes of operation *RESUMEAFTERSTANDSTILL* and *RESUMEAFTERMOTIONSTOP* are equivalent. The superimposed movement would thus also be continued after a restart of the master axis.

Table 1: Overview of the abort conditions for a superimposed movement (MC_MoveSuperimposed)

	ABORTATSTANDSTILL	RESUMEAFTERSTAND-STILL	RESUMEAFTERMO-TIONSTOP
1. Override = 0%	continued	continued	continued
2. Temporary standstill of the main movement	Abort	continued	continued
3. Movement reversal	Abort	continued	continued
4. Axis has reached the target position or is stopped	Abort	Abort	continued
5. Axis reset or switch-off of the enable signal	Abort	Abort	Abort
6. In the case of slave axes: Uncoupling	Abort	Abort	Abort

8.2.6 Data type E_JogMode

The data type E_JogMode is used in conjunction with the function block MC_Jog [▶ 91].

```
TYPE E_JogMode :
(
MC_JOGMODE_STANDARD_SLOW, (* motion with standard jog parameters for slow motion *)
MC_JOGMODE_STANDARD_FAST, (* motion with standard jog parameters for fast motion *)
MC_JOGMODE_CONTINOUS,     (* axis moves as long as the jog button is pressed using parameterized
dynamics *)
MC_JOGMODE_INCHING,       (* axis moves for a certain relative distance *)
MC_JOGMODE_INCHING_MODULO (* axis moves for a certain relative distance - stop position is rounded
to the distance value *)
);
END_TYPE
```

8.3 Status and parameter

8.3.1 Data type E_ReadMode

The data type E_ReadMode is used in conjunction with the function blocks MC_ReadBoolParameter [▶ 27] and MC_ReadBoolParameter [▶ 26] to specify one-off or cyclic mode.

```
TYPE E_ReadMode :
(
READMODE_ONCE := 1,
READMODE_CYCLIC
);
END_TYPE
```

8.3.2 Data type ST_AxisStatus

The data type *ST_AxisStatus* contains comprehensive status information for an axis. The data structure must be updated during each PLC cycle by calling MC_ReadStatus [▶ 30] or by calling the action Axis.ReadStatus (AXIS_REF [▶ 101]).

```
TYPE ST_AxisStatus :
STRUCT
UpdateTaskIndex : BYTE; (* Task-Index of the task that updated this data set *)
UpdateCycleTime : LREAL; (* task cycle time of the task which calls the status function *)
CycleCounter          : UDINT; (* PLC cycle counter when this data set updated *)
NcCycleCounter        : UDINT; (* NC cycle counter incremented after NC task updated NcToPlc data
structures *)

MotionState           : MC_AxisStates; (* motion state in the PLCopen state diagram *)

Error                 : BOOL;  (* axis error state *)
ErrorId               : UDINT; (* axis error code *)

(* PLCopen motion control statemachine states: *)
ErrorStop             : BOOL;
Disabled              : BOOL;
Stopping              : BOOL;
StandStill            : BOOL;
DiscreteMotion        : BOOL;
ContinuousMotion      : BOOL;
SynchronizedMotion    : BOOL;
Homing                : BOOL;

(* additional status - (PLCopen definition)*)
ConstantVelocity      : BOOL;
Accelerating          : BOOL;
Decelerating          : BOOL;

(* Axis.NcToPlc.StateDWord *)
Operational           : BOOL;
ControlLoopClosed     : BOOL; (* operational and position control active *)
HasJob                : BOOL;
HasBeenStopped        : BOOL;
NewTargetPosition     : BOOL; (* new target position commanded during move *)
InPositionArea        : BOOL;
InTargetPosition      : BOOL;
Protected             : BOOL;
Homed                 : BOOL;
HomingBusy            : BOOL;
MotionCommandsLocked  : BOOL; (* stop 'n hold *)
SoftLimitMinExceeded  : BOOL; (* reverse soft travel limit exceeded *)
SoftLimitMaxExceeded  : BOOL; (* forward soft travel limit exceeded *)

Moving                : BOOL;
PositiveDirection     : BOOL;
NegativeDirection     : BOOL;
NotMoving             : BOOL;
Compensating          : BOOL; (* superposition - overlayed motion *)

ExtSetPointGenEnabled : BOOL;
ExternalLatchValid    : BOOL;
CamDataQueued         : BOOL;
CamTableQueued        : BOOL;
CamScalingPending     : BOOL;
CmdBuffered           : BOOL;
PTPmode               : BOOL;
DriveDeviceError      : BOOL;
IoDataInvalid         : BOOL;

(* Axis.NcToPlc.CoupleState *)
Coupled               : BOOL;

(* axis operation mode feedback from NcToPlc *)
OpMode                : ST_AxisOpModes;
END_STRUCT
END_TYPE
```

ST_AxisOpModes [▶ 113] MC_AxisStates [▶ 113]

8.3.3 Data type MC_AxisParameter

The *MC_AxisParameter* data type is used in conjunction with function blocks for reading and writing of axis parameters.

```
TYPE MC_AxisParameter : (
(* PLCopen specific parameters *) (* Index-Group 0x4000 + ID*)
CommandedPosition := 1,           (* lreal *) (* taken from NcToPlc *)
SWLimitPos,                       (* lreal *) (* IndexOffset= 16#0001_000E *)
SWLimitNeg,                       (* lreal *) (* IndexOffset= 16#0001_000D *)
EnableLimitPos,                   (* bool  *) (* IndexOffset= 16#0001_000C *)
EnableLimitNeg,                   (* bool  *) (* IndexOffset= 16#0001_000B *)
EnablePosLagMonitoring,           (* bool  *) (* IndexOffset= 16#0002_0010 *)
MaxPositionLag,                   (* lreal *) (* IndexOffset= 16#0002_0012 *)
MaxVelocitySystem,                (* lreal *) (* IndexOffset= 16#0000_0027 *)
MaxVelocityAppl,                  (* lreal *) (* IndexOffset= 16#0000_0027 *)
ActualVelocity,                   (* lreal *) (* taken from NcToPlc *)
CommandedVelocity,                (* lreal *) (* taken from NcToPlc *)
MaxAccelerationSystem,            (* lreal *) (* IndexOffset= 16#0000_0101 *)
MaxAccelerationAppl,              (* lreal *) (* IndexOffset= 16#0000_0101 *)
MaxDecelerationSystem,            (* lreal *) (* IndexOffset= 16#0000_0102 *)
MaxDecelerationAppl,              (* lreal *) (* IndexOffset= 16#0000_0102 *)
MaxJerkSystem,                    (* lreal *) (* IndexOffset= 16#0000_0103 *)
MaxJerkAppl,                      (* lreal *) (* IndexOffset= 16#0000_0103 *)

(* Beckhoff specific parameters *) (* Index-Group 0x4000 + ID*)
AxisId := 1000,                   (* lreal *) (* IndexOffset= 16#0000_0001 *)
AxisVeloManSlow,                  (* lreal *) (* IndexOffset= 16#0000_0008 *)
AxisVeloManFast,                  (* lreal *) (* IndexOffset= 16#0000_0009 *)
AxisVeloMax,                      (* lreal *) (* IndexOffset= 16#0000_0027 *)
AxisAcc,                          (* lreal *) (* IndexOffset= 16#0000_0101 *)
AxisDec,                          (* lreal *) (* IndexOffset= 16#0000_0102 *)
AxisJerk,                         (* lreal *) (* IndexOffset= 16#0000_0103 *)
MaxJerk,                          (* lreal *) (* IndexOffset= 16#0000_0103 *)
AxisMaxVelocity,                  (* lreal *) (* IndexOffset= 16#0000_0027 *)
AxisRapidTraverseVelocity,        (* lreal *) (* IndexOffset= 16#0000_000A *)
AxisManualVelocityFast,           (* lreal *) (* IndexOffset= 16#0000_0009 *)
AxisManualVelocitySlow,           (* lreal *) (* IndexOffset= 16#0000_0008 *)
AxisCalibrationVelocityForward,   (* lreal *) (* IndexOffset= 16#0000_0006 *)
AxisCalibrationVelocityBackward,  (* lreal *) (* IndexOffset= 16#0000_0007 *)
AxisJogIncrementForward,          (* lreal *) (* IndexOffset= 16#0000_0018 *)
AxisJogIncrementBackward,         (* lreal *) (* IndexOffset= 16#0000_0019 *)
AxisEnMinSoftPosLimit,            (* bool  *) (* IndexOffset= 16#0001_000B *)
AxisMinSoftPosLimit,              (* lreal *) (* IndexOffset= 16#0001_000D *)
AxisEnMaxSoftPosLimit,            (* bool  *) (* IndexOffset= 16#0001_000C *)
AxisMaxSoftPosLimit,              (* lreal *) (* IndexOffset= 16#0001_000E *)
AxisEnPositionLagMonitoring,      (* bool  *) (* IndexOffset= 16#0002_0010 *)
AxisMaxPosLagValue,               (* lreal *) (* IndexOffset= 16#0002_0012 *)
AxisMaxPosLagFilterTime,          (* lreal *) (* IndexOffset= 16#0002_0013 *)
AxisEnPositionRangeMonitoring,    (* bool  *) (* IndexOffset= 16#0000_000F *)
AxisPositionRangeWindow,          (* lreal *) (* IndexOffset= 16#0000_0010 *)
AxisEnTargetPositionMonitoring,   (* bool  *) (* IndexOffset= 16#0000_0015 *)
AxisTargetPositionWindow,         (* lreal *) (* IndexOffset= 16#0000_0016 *)
AxisTargetPositionMonitoringTime, (* lreal *) (* IndexOffset= 16#0000_0017 *)
AxisEnInTargetTimeout,            (* bool  *) (* IndexOffset= 16#0000_0029 *)
AxisInTargetTimeout,              (* lreal *) (* IndexOffset= 16#0000_002A *)
AxisEnMotionMonitoring,           (* bool  *) (* IndexOffset= 16#0000_0011 *)
AxisMotionMonitoringWindow,       (* lreal *) (* IndexOffset= 16#0000_0028 *)
AxisMotionMonitoringTime,         (* lreal *) (* IndexOffset= 16#0000_0012 *)
AxisDelayTimeVeloPosition,        (* lreal *) (* IndexOffset= 16#0000_0104 *)
AxisEnLoopingDistance,            (* bool  *) (* IndexOffset= 16#0000_0013 *)
AxisLoopingDistance,              (* lreal *) (* IndexOffset= 16#0000_0014 *)
AxisEnBacklashCompensation,       (* bool  *) (* IndexOffset= 16#0000_002B *)
AxisBacklash,                     (* lreal *) (* IndexOffset= 16#0000_002C *)
AxisEnDataPersistence,            (* bool  *) (* IndexOffset= 16#0000_0030 *)
AxisRefVeloOnRefOutput,           (* lreal *) (* IndexOffset= 16#0003_0101 *)
AxisOverrideType,                 (* lreal *) (* IndexOffset= 16#0000_0105 *)
(* new since 4/2007 *)
AxisEncoderScalingFactor,         (* lreal *) (* IndexOffset= 16#0001_0006 *)
AxisEncoderOffset,                (* lreal *) (* IndexOffset= 16#0001_0007 *)
AxisEncoderDirectionInverse,      (* bool  *) (* IndexOffset= 16#0001_0008 *)
AxisEncoderMask,                  (* dword *) (* IndexOffset= 16#0001_0015 *)
AxisEncoderModuloValue,           (* lreal *) (* IndexOffset= 16#0001_0009 *)
AxisModuloToleranceWindow,        (* lreal *) (* IndexOffset= 16#0001_001B *)
AxisEnablePosCorrection,          (* bool  *) (* IndexOffset= 16#0001_0016 *)
AxisPosCorrectionFilterTime,      (* lreal *) (* IndexOffset= 16#0001_0017 *)
(* new since 1/2010 *)
AxisUnitInterpretation,           (* lreal *) (* IndexOffset= 16#0000_0026 *)
```

```
AxisMotorDirectionInverse,        (* bool  *) (* IndexOffset= 16#0003_0006 *)
(* new since 1/2011 *)
AxisCycleTime,                    (* lreal *) (* IndexOffset= 16#0000_0004 *)
(* new since 5/2011 *)
AxisFastStopSignalType,           (* dword *) (* IndexOffset= 16#0000_001E *)
AxisFastAcc,                      (* lreal *) (* IndexOffset= 16#0000_010A *)
AxisFastDec,                      (* lreal *) (* IndexOffset= 16#0000_010B *)
AxisFastJerk,                     (* lreal *) (* IndexOffset= 16#0000_010C *)

(* Beckhoff specific axis status information - READ ONLY *) (* Index-Group 0x4100 + ID*)
AxisTargetPosition := 2000,       (* lreal *) (* IndexOffset= 16#0000_0013 *)
AxisRemainingTimeToGo,            (* lreal *) (* IndexOffset= 16#0000_0014 *)
AxisRemainingDistanceToGo,        (* lreal *) (* IndexOffset= 16#0000_0022, 16#0000_0042 *)

(* Beckhoff specific axis functions *)
(* read/write gear ratio of a slave *)
AxisGearRatio := 3000,            (* lreal *) (* read: IndexGroup=0x4100+ID, IdxOffset=16#0000_0022,
*)
(* write:IndexGroup=0x4200+ID, IdxOffset=16#0000_0042 *)

(* Beckhoff specific other parameters *)
(* new since 1/2011 *)
NcSafCycleTime := 4000,           (* lreal *) (* IndexOffset= 16#0000_0010 *)
NcSvbCycleTime                    (* lreal *) (* IndexOffset= 16#0000_0012 *)
);
END_TYPE
```

The *AxisGearRatio* parameter can only be read or written if the axis is coupled as a slave. During the motion only very small changes are allowed.

8.3.4 Data type ST_PowerStepperStruct

```
TYPE ST_PowerStepperStruct :
STRUCT
DestallDetectMode : E_DestallDetectMode;
DestallMode : E_DestallMode;
DestallEnable : BOOL;
StatusMonEnable : BOOL;
Retries : INT;
Timeout : TIME;
END_STRUCT
END_TYPE
```

8.3.5 Data type ST_DriveAddress

The data type *ST_DriveAddress* contains the ADS access data for a drive unit. The data are read with MC_ReadDriveAddress [▶ 55].

```
TYPE ST_DriveAddress :
STRUCT
NetID        : T_AmsNetId;     (* AMS NetID of the drive as a string *)
NetIdBytes   : T_AmsNetIdArr;  (* AMS NetID of the drive as a byte array (same information as NetID)
*)
SlaveAddress : T_AmsPort;      (* slave address of the drive connected to a bus master *)
Channel      : BYTE;           (* EtherCAT channel number of the drive (0, 1, 2, 3, 4…) *)
END_STRUCT
END_TYPE
```

8.3.6 Data type ST_AxisParameterSet

The data type *ST_AxisParameterSet* contains the whole parameter dataset of an axis that can be read with the function block MC_ReadParameterSet [▶ 29].

Individual parameters that can be changed at runtime can be written with MC_WriteParameter [▶ 33]. It is not possible to write back the parameter dataset.

The individual parameters are described in the NC ADS documentation.

```
TYPE ST_AxisParameterSet :
STRUCT
(* AXIS: *)
AxisId                       : DWORD; (* 0x00000001 *)
sAxisName                    : STRING(31); (* 0x00000002 *)
nAxisType                    : DWORD;      (* 0x00000003 *)
bEnablePositionAreaControl   : WORD;       (* 0x0000000F *)
fPositionAreaControlRange    : LREAL;      (* 0x00000010 *)
bEnableMotionControl         : WORD;       (* 0x00000011 *)
fMotionControlTime           : LREAL;      (* 0x00000012 *)
bEnableLoop                  : WORD;       (* 0x00000013 *)
fLoopDistance                : LREAL;      (* 0x00000014 *)
bEnableTargetPosControl      : WORD;       (* 0x00000015 *)
fTargetPosControlRange       : LREAL;      (* 0x00000016 *)
fTargetPosControlTime        : LREAL;      (* 0x00000017 *)
fVeloMaximum                 : LREAL;      (* 0x00000027 *)
fMotionControlRange          : LREAL;      (* 0x00000028 *)
bEnablePEHTimeControl        : WORD;       (* 0x00000029 *)
fPEHControlTime              : LREAL;      (* 0x0000002A *)
bEnableBacklashCompensation  : WORD;       (* 0x0000002B *)
fBacklash                    : LREAL;      (* 0x0000002C *)
sAmsNetId                    : T_AmsNetId; (* 0x00000031 *)
nPort                        : WORD;       (* 0x00000031 *)
nChnNo                       : WORD;       (* 0x00000031 *)
fAcceleration                : LREAL;      (* 0x00000101 *)
fDeceleration                : LREAL;      (* 0x00000102 *)
fJerk                        : LREAL;      (* 0x00000103 *)

(* ENCODER: *)
nEncId                       : DWORD;      (* 0x00010001 *)
sEncName                     : STRING(31); (* 0x00010002 *)
nEncType                     : DWORD;      (* 0x00010003 *)
fEncScaleFactor              : LREAL;      (* 0x00010006 *)
fEncOffset                   : LREAL;      (* 0x00010007 *)
bEncIsInverse                : WORD;       (* 0x00010008 *)
fEncModuloFactor             : LREAL;      (* 0x00010009 *)
nEncMode                     : DWORD;      (* 0x0001000A *)
bEncEnableSoftEndMinControl  : WORD;       (* 0x0001000B *)
bEncEnableSoftEndMaxControl  : WORD;       (* 0x0001000C *)
fEncSoftEndMin               : LREAL;      (* 0x0001000D *)
fEncSoftEndMax               : LREAL;      (* 0x0001000E *)
nEncMaxIncrement             : DWORD;      (* 0x00010015 *)
bEncEnablePosCorrection      : WORD;       (* 0x00010016 *)
fEncPosCorrectionFilterTime  : LREAL;      (* 0x00010017 *)

(* CONTROLLER: *)
nCtrlId                      : DWORD;      (* 0x00020001 *)
sCtrlName                    : STRING(31); (* 0x00020002 *)
nCtrlType                    : DWORD;      (* 0x00020003 *)
bCtrlEnablePosDiffControl    : WORD;       (* 0x00020010 *)
bCtrlEnableVeloDiffControl   : WORD;       (* 0x00020011 *)
fCtrlPosDiffMax              : LREAL;      (* 0x00020012 *)
fCtrlPosDiffMaxTime          : LREAL;      (* 0x00020013 *)
fCtrlPosKp                   : LREAL;      (* 0x00020102 *)
fCtrlPosTn                   : LREAL;      (* 0x00020103 *)
fCtrlPosTv                   : LREAL;      (* 0x00020104 *)
fCtrlPosTd                   : LREAL;      (* 0x00020105 *)
fCtrlPosExtKp                : LREAL;      (* 0x00020106 *)
fCtrlPosExtVelo              : LREAL;      (* 0x00020107 *)
fCtrlAccKa                   : LREAL;      (* 0x00020108 *)

(* DRIVE: *)
nDriveId                     : DWORD;      (* 0x00030001 *)
sDriveName                   : STRING(31); (* 0x00030002 *)
nDriveType                   : DWORD;      (* 0x00030003 *)
bDriveIsInverse              : WORD;       (* 0x00030006 *)
fDriveVeloReferenz           : LREAL;      (* 0x00030101 *)
fDriveOutputReferenz         : LREAL;      (* 0x00030102 *)

(* miscellaneous *)
fAxisCycleTime               : LREAL;      (* 0x00000004 *) (* available from Tc 2.11 R2 *)

(* 17.05.11: parameter extension *)
fRefVeloSearch               : LREAL;      (* 0x00000006 calibration velo (TO plc cam) *)
fRefVeloSync                 : LREAL;      (* 0x00000007 calibration velo (off plc cam) *)
fVeloSlowManual              : LREAL;      (* 0x00000008 manual velocity (slow) *)
fVeloFastManual              : LREAL;      (* 0x00000009 manual velocity (fast) *)
```

```
(* ENCODER (incremental): *)
bEncRefSearchInverse        : UINT;       (* 0x00010101 *)
bEncRefSyncInverse          : UINT;       (* 0x00010102 *)
nEncRefMode                 : UDINT;      (* 0x00010107 *)
fEncRefPosition             : LREAL;      (* 0x00010103 *)

(* fill up *)
arrReserved                 : ARRAY[511..512] OF BYTE; (* fill up to 512 bytes *)
END_STRUCT
END_TYPE
```

8.3.7 Data type ST_AxisOpModes

The data type *ST_AxisOpModes* contains information about the operating mode parameterization of an axis.

```
TYPE ST_AxisOpModes :
STRUCT
PositionAreaMonitoring  : BOOL; (* bit 0 - OpModeDWord *)
TargetPositionMonitoring: BOOL; (* bit 1 - OpModeDWord *)
LoopMode                : BOOL; (* bit 2 - OpModeDWord - loop mode for two speed axes *)
MotionMonitoring        : BOOL; (* bit 3 - OpModeDWord *)
PEHTimeMonitoring       : BOOL; (* bit 4 - OpModeDWord *)
BacklashCompensation    : BOOL; (* bit 5 - OpModeDWord *)
Modulo                  : BOOL; (* bit 7 - OpModeDWord - axis is parameterized as modulo axis *)
PositionLagMonitoring   : BOOL; (* bit 16 - OpModeDWord *)
VelocityLagMonitoring   : BOOL; (* bit 17 - OpModeDWord *)
SoftLimitMinMonitoring  : BOOL; (* bit 18 - OpModeDWord *)
SoftLimitMaxMonitoring  : BOOL; (* bit 19 - OpModeDWord *)
PositionCorrection : BOOL; (* bit 20 - OpModeDWord *)
END_STRUCT
END_TYPE
```

8.3.8 Data type E_AxisPositionCorrectionMode

```
TYPE E_PositionCorrectionMode:
(
POSITIONCORRECTION_MODE_UNLIMITED, (* no limitation - pass correction immediately *)
POSITIONCORRECTION_MODE_FAST,      (* limitatation to maximum position change per cycle *)
POSITIONCORRECTION_MODE_FULLLENGTH (* limitation uses full length to adapt to correction in small
steps *)
);
END_TYPE
```

POSITIONCORRECTION_MODE_UNLIMITED	No filtering, the correction is executed immediately. Note that large changes in the correction value can lead to high accelerations.
POSITIONCORRECTION_MODE_FAST	The position correction is limited to the extent that a maximum acceleration is not exceeded. However, the correction is completely executed as fast as possible.
POSITIONCORRECTION_MODE_FULLLENGTH	The position correction is accomplished distributed over a distance of the axis (*CorrectionLength*). This results in smaller changes per time unit.

8.3.9 Data type MC_AxisStates

The data type *MC_AxisStates* describes the operating states according to the PlcOpen state diagram [▶ 11].

```
TYPE MC_AxisStates :
(
MC_AXISSTATE_UNDEFINED,
MC_AXISSTATE_DISABLED,
MC_AXISSTATE_STANDSTILL,
MC_AXISSTATE_ERRORSTOP,
MC_AXISSTATE_STOPPING,
MC_AXISSTATE_HOMING,
MC_AXISSTATE_DISCRETEMOTION,
MC_AXISSTATE_CONTINOUSMOTION,
MC_AXISSTATE_SYNCHRONIZEDMOTION
```

```
);
END_TYPE
```

See also: General rules for MC function blocks [▶ 14]

8.4 Touch probe

8.4.1 Data type TRIGGER_REF

```
TYPE TRIGGER_REF :
STRUCT
EncoderID       : UDINT;              (* 1..255 *)
TouchProbe      : E_TouchProbe;       (* probe unit definition *)
SignalSource    : E_SignalSource;     (* optional physical signal source used by the probe unit
                                         - available from TwinCAT 2.11 Build 2022 with
MC_TouchProbe_V2 *)
Edge            : E_SignalEdge;       (* rising or falling signal edge *)
Mode            : E_TouchProbeMode;   (* single shot or continuous monitoring
                                         - available from TwinCAT 2.11 Build 2022 with
MC_TouchProbe_V2 *)
PlcEvent        : BOOL;               (* PLC trigger signal input when TouchProbe signal source is
set to 'PlcEvent' *)
ModuloPositions : BOOL;               (* interpretation of FirstPosition, LastPosition and
RecordedPosition as modulo positions when TRUE *)
END_STRUCT
END_TYPE
```

EncoderID: The ID of an encoder is indicated in the TwinCAT System Manager.

TouchProbe : Defines the latch unit (probe unit) within the encoder hardware used.

```
TYPE E_TouchProbe :
(
TouchProbe1 := 1,     (* 1st hardware probe unit with Sercos, CanOpen, KL5xxx and others *)
TouchProbe2,          (* 2nd probe unit - available from MC_TouchProbe_V2_00 *)
TouchProbe3,          (* currently not available *)
TouchProbe4,          (* currently not available *)
PlcEvent := 10        (* simple PLC signal TRUE/FALSE *)
);
END_TYPE
```

SignalSource: Optionally defines the signal source, if it can be selected via the controller. In many cases the signal source is permanently configured in the drive and should then be set to the default value *SignalSource_Default*. (setting possibility only available from MC_TouchProbe_V2 [▶ 37])

```
TYPE E_SignalSource :
(
SignalSource_Default,           (* undefined or externally configured *)
SignalSource_Input1,            (* digital drive input 1 *)
SignalSource_Input2,            (* digital drive input 2 *)
SignalSource_Input3,            (* digital drive input 3 *)
SignalSource_Input4,            (* digital drive input 4 *)
SignalSource_ZeroPulse := 128,  (* encoder zero pulse *)
SignalSource_DriveDefined       (* defined by drive parameters - e. g. CAN object 0x60D0 *)
);
END_TYPE
```

Edge : Defines whether the rising or falling edge of the trigger signal is evaluated.

```
TYPE E_SignalEdge :
(
RisingEdge,
FallingEdge
);
END_TYPE
```

Mode: Specifies the operating mode of the latch unit. In single mode only the first edge is sampled. In continuous mode each PLC cycle edge is signaled. (*Mode* only available with MC_TouchProbe_V2 [▶ 37])

```
TYPE E_TouchProbeMode :
(
TOUCHPROBEMODE_SINGLE_COMPATIBILITYMODE,   (* for TwinCAT 2.10 and 2.11 before Build 2022 - for use
with MC_TouchProbe as well *)
TOUCHPROBEMODE_SINGLE,                     (* multi probe interface - from 2.11 Build 2022 *)
TOUCHPROBEMODE_CONTINOUS                   (* multi probe interface - from 2.11 Build 2022 *)
);
END_TYPE
```

PlcEvent : If the signal source TouchProbe is set to the type PlcEvent, a rising edge on these variables triggers the recording of the current axis position. The PlcEvent is not a true latch function, but is cycle-time dependent.

ModuloPositions: If the variable *ModuloPositions* is FALSE, the axis position is interpreted in an absolute linear range from -∞ to +∞. The positions *FirstPosition*, *LastPosition* und *RecordedPosition* of the MC_TouchProbe [▶ 34] or MC_TouchProbe_V2 [▶ 37] function block are then also absolute.
If ModuloPositions is TRUE, all positions are interpreted as modulo values in the modulo range of the axis used (e.g. 0..359.9999). At the same time this means that a defined trigger window repeats itself cyclically.

8.4.2 Data type MC_TouchProbeRecordedData

```
TYPE MC_TouchProbeRecordedData :
STRUCT
    Counter             : LREAL;
    RecordedPosition    : LREAL;
    AbsolutePosition    : LREAL;
    ModuloPosition      : LREAL;
END_STRUCT
END_TYPE
```

Counter: counter indicating how many valid edges were detected in the last cycle. Detection of multiple edges is only implemented in mode TOUCHPROBEMODE_CONTINUOUS under SERCOS / SOE and must be supported explicitly by the hardware (e.g. AX5000).

RecordedPosition: one or more detected axis positions at the time of the trigger signal. This corresponds to the absolute axis position or the modulo axis position, depending on the parameterization.

AbsolutePosition: absolute axis position detected at the time of the trigger signal.

ModuloPosition: modulo axis position recorded at the time of the trigger signal.

8.5 External set value generator

8.5.1 Datentyp E_PositionType

```
TYPE E_PositionType :
(
POSITIONTYPE_ABSOLUTE  := 1, (* Absolute position *)
POSITIONTYPE_RELATIVE,       (* Relative position *)
POSITIONTYPE_MODULO    := 5  (* Modulo position *)
);
END_TYPE
```

9 Example programs

9.1 Sample Programs

PTP – point to point movement

The example program manages and moves an axis in PTP mode. The axis is moved with two instances of an MC_MoveAbsolute function block in queued mode over several intermediate positions and velocity levels.

The example program requires the TcMC2.lib library and operates fully in simulation mode. Progress can be monitored in TwinCAT Scope View with the configuration provided.

Click here to save the example program:

https://infosys.beckhoff.com/content/1033/tcplclibmc2/Resources/458496267/.zip

Master-Slave coupling

The example program couples two axes and moves them together. The slave axis is uncoupled and positioned during the journey.

https://infosys.beckhoff.com/content/1033/tcplclibmc2/Resources/458499211/.zip

Dancer control

The dancer control example program shows how the speed of a slave axis can be controlled using the position of a dancer.

https://infosys.beckhoff.com/content/1033/tcplclibmc2/Resources/458502155/.zip

Superimposed movement (Superposition)

The example shows the overlay of a movement while an axis is driving.

https://infosys.beckhoff.com/content/1033/tcplclibmc2/Resources/458505099/.zip

More Information:
www.beckhoff.com/tx1200

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Germany
Phone: +49 5246 9630
info@beckhoff.com
www.beckhoff.com