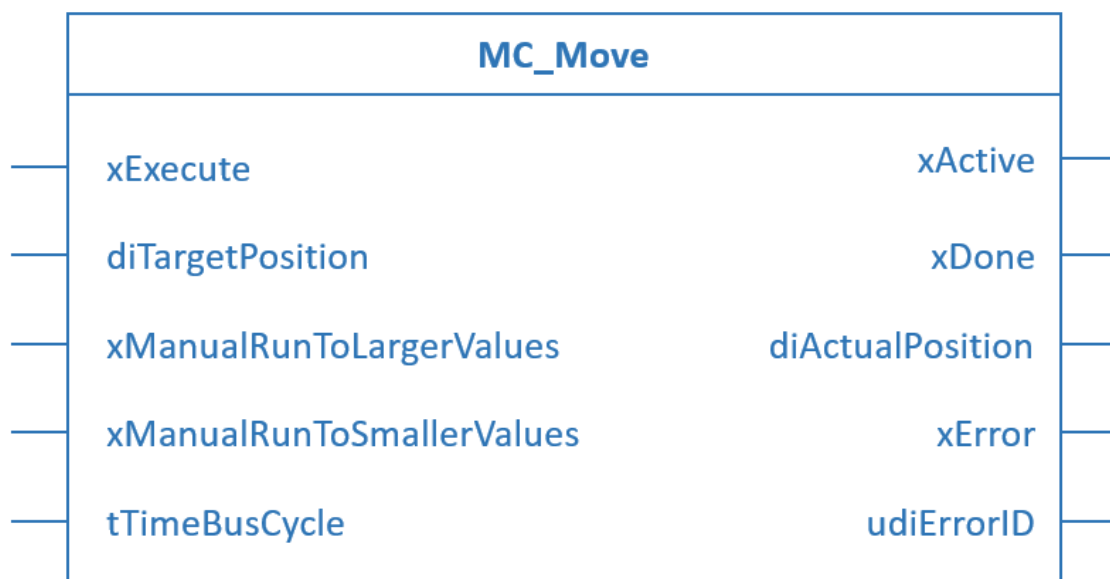


Function Blocks PSx3xx for TIA-Portal

- SIMATIC S7 - 300
- SIMATIC S7 - 1200 und SIMATIC S7 - 1500



Purpose of instruction manual

This instruction manual describes the function blocks for the PSx3xx PN (with PROFINET interface).

Improper use of these devices or failure to follow these instructions may cause injury or equipment damage. Every person who uses the devices must therefore read the instruction manual and has to understand the possible risks. The instruction manual and in particular the safety precautions, contained therein, must be followed carefully.

Contact them manufacturer if you do not understand any part of this instruction manual.

The manufacturer reserves the right to continue developing these function blocks without documenting such development in each individual case. The manufacturer will be happy to determine whether this manual is up-to-date.

The copyright to these operating instructions is with the manufacturer. It contains technical data, instructions and drawings for the function and handling of the software. It is forbidden to be reproduced or made accessible to third parties in whole or in part.

halstrup-walcher GmbH
Stegener Straße 10
79199 Kirchzarten

Tel. +49 (7661) 39 63-0
info@halstrup-walcher.de
www.halstrup-walcher.de

© 2023

09.11.2023, TS & JB

7100.005454G Version 2.1 Instruction Manual Function Blocks PSx3xx PN

Table of contents

1	Safety precautions	5
1.1	Appropriate use	5
1.2	Symbols	5
2	User-specific data types.....	6
2.1	<ST_DriveData_PSx3xx>	6
2.2	<ST_FunctionBlocks_PSx3xx>	7
2.3	<ST_Variables_PSx3xx>	7
3	<GVL_Data_Store_PSx3xx>.....	8
4	Error description (<udiErrorID>)	9
5	Description of the function blocks.....	11
5.1	FC_MC_Communication_PSx3xx_S7-1200_1500 (FC101), FC_MC_Communication_PSx3xx_S7-300 (FC101)	11
5.2	FB_MC_Move_PSx3xx.....	11
5.3	FB_MC_Error_PSx3xx	12
5.4	FC_MC_Error_ID_PSx3xx	12
5.5	FB_MC_ReadParameter_PSx3xx	12
5.6	FB_MC_WriteParameter_PSx3xx.....	13
5.7	FB_MC_Parametrization_PSx3xx	13
5.8	FB_MC_PosParametrization_PSx3xx.....	14
6	Description of the parameters	15
6.1	<xActive>	15
6.2	<diActualPosition>	15
6.3	<xDeliveryState>	16
6.4	<uiDirection>	16
6.5	<xDone>	16
6.6	<stDrive>	17
6.7	<xError>	17
6.8	<sErrorDescription>	17
6.9	<udiErrorID>	18
6.10	<uiErrorParameter>.....	18
6.11	<xExecute>	19
6.12	<uiInputAddress>	19
6.13	<diLowerLimit>.....	20
6.14	<xManualRunToLargerValues>	20
6.15	<xManualRunToSmallerValues>	20
6.16	<uiOutputAddress>	21

6.17	<stParameter>	21
6.18	<uiParameterNumber>	21
6.19	<xSaveSettings>	22
6.20	<diSetPoint>	22
6.21	<uiStepsPerTurn>	22
6.22	<usiSubIndex>	22
6.23	<diTargetPosition>	23
6.24	<tTimeBusCycle>	23
6.25	<diUpperLimit>	24
6.26	<diValue>	24
6.27	<sValue>	24
7	Application of the function blocks	25
7.1	Project	25
7.2	Library	25
7.3	Interlock between the function blocks	26
7.4	Several PSx3xx in a project	27
8	Example programs	28
8.1	Example 1: positioning mode	28
8.2	Example 2: manual mode	29
8.3	Example 3: read actual value	29
8.4	Example 4: writing the delivery state	30
8.5	Example 5: Parameterize parameters	30
8.6	Example 6: Parameterize position parameters	31
8.7	Example 7: Error upper end limit reached	31
	List of Figures	32

1 Safety precautions

1.1 Appropriate use

The positioning systems PSx3xx PN are especially suitable for automatically setting tools, stops or spindles for wood-processing equipment, packing lines, printing equipment, filling units and other types of special machines.

PSx3xx PN positioning systems are not stand-alone devices and may only be used, if coupled to another machine.

1.2 Symbols

The symbols given below are used throughout this instruction manual to indicate instances when improper operation could result in the following hazards:



WARNING!

If the corresponding instructions are not followed, this warns you of a potential hazard that could lead to bodily injury up to and including death.



CAUTION!

If corresponding instructions are not followed, this alerts you of a potential hazard that could lead to significant property damage.



INFORMATION!

This indicates that the corresponding information is important for operating the function blocks properly.

2 User-specific data types

There are many user-specific data types. Only the three most important data types are documented below.

2.1 <ST_DriveData_PSx3xx>

A data structure is used to store some of the drives' data. For each drive, a global instance of this structure is required. This instance must be provided to each function block (FB) that operates on the corresponding drive. Here, it will be prevented, that two FBs can access the parameter interface of a single drive. Furthermore, in this data structure the addresses for the input and output data of the corresponding drive has to be deposited.

Parameter name	Data type	Written by	Description
<uiPdAdressIn>	UInt (S7-1200 und S7-1500) DWord (S7-300)	User	Address process data (Device → Controller)
<uiPdAddressOut>	UInt (S7-1200 und S7-1500) DWord (S7-300)	User	Address process data (Controller → Device)
<sAxisName>	String[16]	User (optional)	Name of axis
<sAxisDescription>	String[32]	User (optional)	Description (e. g. function, task of this axis)
<iActiveFunktionBlock>	Int	Function blocks	Active function block
<xCommunicationError>	Bool	Function blocks	Communication error to IO-Device
<stPrivate>	ST_Private	Function blocks	Data structure for internal use

The following illustrations show how the assigned addresses of the master and the slave can be checked in the TIA-Portal automation software:

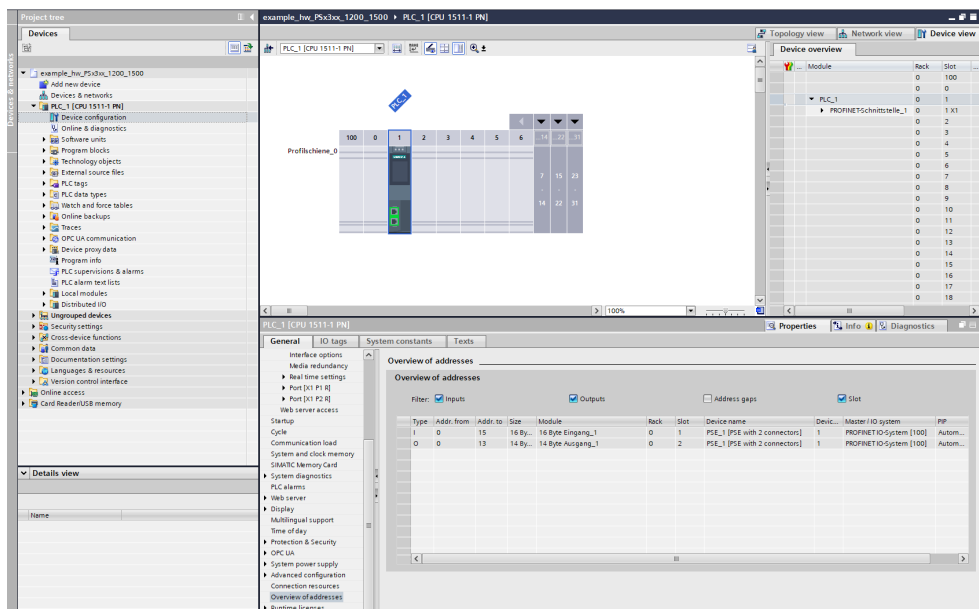


Figure 1: Addresses of the process data from the perspective of the PLC

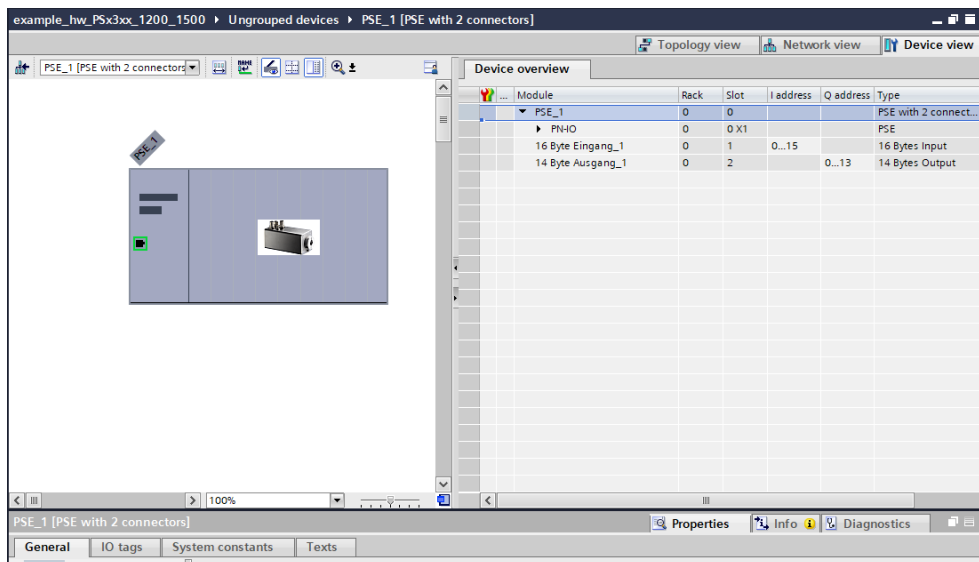


Figure 2: Addresses to be set from the perspective of the device

2.2 <ST_FunctionBlocks_PSx3xx>

For each drive there is a data structure in which the variables for the assignment to the function blocks are stored. This structure has several substructures, which are not explained further in the documentation. A global instance of this structure is required for each drive. If function blocks are not used, it is recommended to adapt the data structure so that memory is not unnecessarily occupied in the PLC.

Parameter name	Data type	Written by	Description
<stMove_PSx3xx>	ST_Move_PSx3xx	Function blocks	variable for FB_MC_Move_PSx3xx
<stError_PSx3xx>	ST_Error_PSx3xx	Function blocks	variable for FB_MC_Error_PSx3xx
<stReadParameter_PSx3xx>	ST_ReadParameter_PSx3xx	Function blocks	variable for FB_MC_ReadParameter_PSx3xx
<stWriteParameter_PSx3xx>	ST_WriteParameter_PSx3xx	Function blocks	variable for FB_MC_WriteParameter_PSx3xx
<stParametrization_PSx3xx>	ST_Parametrization_PSx3xx	Function blocks	variable for FB_MC_Parametrization_PSx3xx
<stPosParametrization_PSx3xx>	ST_PosParametrization_PSx3xx	Function blocks	variable for FB_MC_PosParametrization_PSx3xx

2.3 <ST_Variables_PSx3xx>

In this data structure the structures <ST_FunctionBlocks_PSx3xx> and <ST_DriveData_PSx3xx> are called. This structure is transferred to each drive in the <GVL_Data_Store_PSx3xx> global variable list.

3 <GVL_Data_Store_PSx3xx>

The GVL (global variable list) <GVL_Data_Store_PSx3xx> provides the necessary variables to assign all inputs and outputs of all available function blocks.

Each variable is currently only available once. If there are several PSx3xx in a project, the number of variables must be multiplied accordingly.

Example

```

25 // "FB_MC_Move_PSx3xx_DB" --> instance of "FB_MC_Move_PSx3xx"
26 "FB_MC_Move_PSx3xx_DB" (xExecute := "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stMove_PSx3xx.stInput_Move_PSx3xx.xExecute,
27   diTargetPosition := "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stMove_PSx3xx.stInput_Move_PSx3xx.diTargetPosition,
28   xManualRunToLargerValues := "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stMove_PSx3xx.stInput_Move_PSx3xx.xManualRunToLargerValues,
29   xManualRunToSmallerValues := "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stMove_PSx3xx.stInput_Move_PSx3xx.xManualRunToSmallerValues,
30   tTimeBusCycle := "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stMove_PSx3xx.stInput_Move_PSx3xx.tTimeBusCycle,
31   xActive => "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stMove_PSx3xx.stOutput_Move_PSx3xx.xActive,
32   xDone => "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stMove_PSx3xx.stOutput_Move_PSx3xx.xDone,
33   diActualPosition => "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stMove_PSx3xx.stOutput_Move_PSx3xx.diActualPosition,
34   xError => "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stMove_PSx3xx.stOutput_Move_PSx3xx.xError,
35   udiErrorID => "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stMove_PSx3xx.stOutput_Move_PSx3xx.udiErrorID,
36   stDrive := "GVL_Data_Store_PSx3xx".PSE_1.stDriveData_PSx3xx);
37

```

Figure 3: Allocation of the variables from <GVL_Data_Store_PSx3xx> to FB_MC_Move_PSx3xx

If the GVL is transferred to the project, it is recommended to delete the unused variables (so that memory is not unnecessarily occupied in the PLC) and to adapt the block to the number of drives actually present.

example_hw_PSx3xx_S7_1200_1500 ▶ PLC_1 [CPU 1511-1 PN] ▶ Program blocks ▶ GVLs ▶ GVLs_halstrup_walcher ▶ GVL_Data_Store_PSx3xx [DB100]

<

Figure 4: <GVL_Data_Store_PSx3xx> with the variables for a drive

4 Error description (<udiErrorID>)

The error codes output by the function blocks are described below:

<udiErrorID> (hex)	Description
16#F0000 (Mask)	Function block
16#0xxxx	No error code
16#1xxxx	Error in FB_MC_Move_PSx3xx
16#2xxxx	Error in FB_MC_Error_PSx3xx
16#3xxxx	Error in FB_MC_ReadParameter_PSx3xx
16#4xxxx	Error in FB_MC_WriteParameter_PSx3xx
16#5xxxx	Error in FB_MC_Parametrization_PSx3xx
16#6xxxx	Error in FB_MC_PosParametrization_PSx3xx
16#0F000 (Mask)	Internal errors in the function blocks and process data errors
16#x0xxx	No internal error in the function blocks and process data error
16#x1xxx	Error in the state machine (interlock)
16#x2xxx	Invalid process data input address
16#x3xxx	Invalid process data output address
16#x4xxx	Error while reading process data
16#x5xxx	Error while writing process data
16#x6xxx	Invalid write command
16#00F00 (Mask)	Error in parameter
16#xx0xx	No parameter errors
16#xx1xx	Parameter: communication timeout (1000 ms)
16#xx2xx	Parameter: invalid parameter number
16#xx3xx	Parameter: parameter value is read only
16#xx4xx	Parameter: below lower limit or upper limit exceeded
16#xx5xx	Parameter: invalid subindex
16#xx6xx	Parameter: no array
16#xx7xx	Parameter: incorrect data type
16#xx8xx	Parameter: setting not allowed (resetting only)
16#xx9xx	Parameter: request cannot be processed due to operating state
16#xxAxx	Other error
16#000F0 (Mask)	Drive errors
16#xxx0x	No drive errors
16#xxx1x	Drag error
16#xxx2x	Under- or overvoltage motor supply (STO-enabling inactive*)
16#xxx3x	Positioning run aborted
16#xxx4x	Temperature exceeded
16#xxx5x	Measuring system error (Measuring system error or STO hardware error*)
16#xxx6x	Positioning error (block)
16#xxx7x	Manual displacement
16#xxx8x	Incorrect target value
16#xxx9x	Under- or overvoltage during run / failure voltage control
16#xxxAx	Below lower limit
16#xxxBx	Upper limit exceeded

* If the PSx3xx is a device with STO, then the error description in brackets must be taken into account! (<udiErrorID>: 16#xxx2x and 16#xxx5x)

The errors "Drive Errors" are a copy of the error bits in the status word of the PSx3xx.

Examples

- Run command (**FB_MC_Move_PSx3xx**) with incorrect target value →
<udiErrorID> = 16#10080
- Writing a parameter (**FB_MC_WriteParameter_PSx3xx**) with invalid parameter number →
<udiErrorID> = 16#40200

5 Description of the function blocks

A detailed description of all parameters is in chapter 6 Description of the parameters.

5.1 FC_MC_Communication_Ps3xx_S7-1200_1500 (FC101), FC_MC_Communication_Ps3xx_S7-300 (FC101)

Use the appropriate function for the following CPU:

CPU	Function
S7-300 CPU	FC_MC_Communication_Ps3xx_S7-300
S7-1200 CPU S7-1500 CPU	FC_MC_Communication_Ps3xx_S7-1200_1500

The functionality is identical in both cases. The **FC_MC_Communication_Ps3xx_S7-1200_1500** function is described below.

This function (FC) is used for communication with the drive (IO device). It carries out the communication centrally for the following blocks:

- MC_Move_Ps3xx
- MC_ReadParameter_Ps3xx
- MC_WriteParameter_Ps3xx
- MC_Error_Ps3xx
- MC_Parametrization_Ps3xx
- MC_PosParametrization_Ps3xx

The input and output module is stored in the data type <driveData_Ps3xx>. All other blocks can access it via this interface.

```

19 //function MC_Communication -> communication to Ps3xx
20 FC_MC_Communication_Ps3xx_S7-1200_1500(uiInputAddress := "GVL_Data_Store_Ps3xx".PSE_1.stDriveData_Ps3xx.uiPdAddressIn,
21                                         uiOutputAddress := "GVL_Data_Store_Ps3xx".PSE_1.stDriveData_Ps3xx.uiPdAddressOut,
22                                         xCommunicationError := "GVL_Data_Store_Ps3xx".PSE_1.stDriveData_Ps3xx.xCommunicationError,
23                                         stDrive := "GVL_Data_Store_Ps3xx".PSE_1.stDriveData_Ps3xx);

```

*Figure 5: Zuweisung der Variablen von <GVL_Data_Store_Ps3xx> zu der FC
FC_MC_Communication_Ps3xx_1200_1500*

5.2 FB_MC_Move_Ps3xx

This function block is used to move the drive.

```

25 //FB_MC_Move_Ps3xx_DB --> instance of "FB_MC_Move_Ps3xx"
26 FB_MC_Move_Ps3xx_DB(xExecute := "GVL_Data_Store_Ps3xx".PSE_1.stFunctionBlocks_Ps3xx.stMove_Ps3xx.xExecute,
27                     diTargetPosition := "GVL_Data_Store_Ps3xx".PSE_1.stFunctionBlocks_Ps3xx.stMove_Ps3xx.stInput_Move_Ps3xx.xTargetPosition,
28                     xManualRunToLargerValues := "GVL_Data_Store_Ps3xx".PSE_1.stFunctionBlocks_Ps3xx.stMove_Ps3xx.stInput_Move_Ps3xx.xManualRunToLargerValues,
29                     xManualRunToSmallerValues := "GVL_Data_Store_Ps3xx".PSE_1.stFunctionBlocks_Ps3xx.stMove_Ps3xx.stInput_Move_Ps3xx.xManualRunToSmallerValues,
30                     tTimeBusCycle := "GVL_Data_Store_Ps3xx".PSE_1.stFunctionBlocks_Ps3xx.stMove_Ps3xx.stInput_Move_Ps3xx.tTimeBusCycle,
31                     xActive => "GVL_Data_Store_Ps3xx".PSE_1.stFunctionBlocks_Ps3xx.stMove_Ps3xx.stOutput_Move_Ps3xx.xActive,
32                     xDone => "GVL_Data_Store_Ps3xx".PSE_1.stFunctionBlocks_Ps3xx.stMove_Ps3xx.stOutput_Move_Ps3xx.xDone,
33                     diActualPosition => "GVL_Data_Store_Ps3xx".PSE_1.stFunctionBlocks_Ps3xx.stMove_Ps3xx.stOutput_Move_Ps3xx.xActualPosition,
34                     xError => "GVL_Data_Store_Ps3xx".PSE_1.stFunctionBlocks_Ps3xx.stMove_Ps3xx.stOutput_Move_Ps3xx.xError,
35                     udiErrorID => "GVL_Data_Store_Ps3xx".PSE_1.stFunctionBlocks_Ps3xx.stMove_Ps3xx.stOutput_Move_Ps3xx.xdiErrorID,
36                     stDrive := "GVL_Data_Store_Ps3xx".PSE_1.stDriveData_Ps3xx);

```

Figure 6: Allocation of variables from DB <Data_Store> to FB MC_Move_Ps3xx

If the drive reports multiple errors, the <udiErrorID> with the highest priority is shown. This applies to all FBs. This priority corresponds to the order in the following table (highest priority has 16#x1xxx):

<udiErrorID>	Description
16#x1xxx	Error in the state machine (interlock)
16#x2xxx	Invalid process data input address
16#x3xxx	Invalid process data output address
16#x4xxx	Error while reading process data
16#x5xxx	Error while writing process data
16#xxx2x	Under- or overvoltage motor supply (STO-enabling inactive*)
16#xxx4x	Temperature exceeded
16#xxx5x	Measuring system error (Measuring system error or STO hardware error*)
16#xxx8x	Incorrect target value
16#xxx9x	Under- or overvoltage during run / failure voltage control
16#xxx6x	Positioning error (block)
16#xxx7x	Manual displacement
16#xxxAx	Below lower limit
16#xxxBx	Upper limit exceeded
16#xxx3x	Positioning run aborted
16#xxx1x	Drag error

*If the PSx3xx is a device with STO, then the error description in brackets must be taken into account! (<udiErrorID>: 16#xxx2x und 16#xxx5x)

5.3 FB_MC_Error_PSx3xx

This FB reports the state of the drive and the FB as error bit, error code (<udiErrorID>) and as text output.

```

59 // "FB_MC_Error_PSx3xx_DB" --> instance of "FB_MC_Error_PSx3xx"
60 "FB_MC_Error_PSx3xx_DB" (xExecute:= "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stError_PSx3xx.stInput_Error_PSx3xx.xExecute,
61   xError:= "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stError_PSx3xx.stOutput_Error_PSx3xx.xError,
62   udiErrorID:= "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stError_PSx3xx.stOutput_Error_PSx3xx.udiErrorID,
63   sErrorDescription:= "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stError_PSx3xx.stOutput_Error_PSx3xx.sErrorDescription,
64   stDrive:= "GVL_Data_Store_PSx3xx".PSE_1.stDriveData_PSx3xx);

```

Figure 7: Allocation of variables from DB <GVL_Data_Store_PSx3xx> to FB_MC_Error_PSx3xx

5.4 FC_MC_Error_ID_PSx3xx

This function is used internally as a sub function of the FBs **FB_MC_Move_PSx3xx**, **FB_MC_ReadParameter_PSx3xx**, **FB_MC_WriteParameter_PSx3xx** and **FB_MC_Error_PSx3xx**. It just has to be copied from the library into the application program. The setting of the inputs is done completely internally.

5.5 FB_MC_ReadParameter_PSx3xx

This FB can be used to read out parameter values from the drive.

```

39 // "FB_MC_ReadParameter_PSx3xx_DB" --> instance of "FB_MC_ReadParameter_PSx3xx"
39 "FB_MC_ReadParameter_PSx3xx_DB" (xExecute:= "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stReadParameter_PSx3xx.stInput_ReadParameter_PSx3xx.xExecute,
40   uiParameterNumber:= "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stReadParameter_PSx3xx.stInput_ReadParameter_PSx3xx.uiParameterNumber,
41   uiSubindex:= "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stReadParameter_PSx3xx.stInput_ReadParameter_PSx3xx.uiSubindex,
42   xActive:= "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stReadParameter_PSx3xx.stOutput_ReadParameter_PSx3xx.xActive,
43   xDone:= "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stReadParameter_PSx3xx.stOutput_ReadParameter_PSx3xx.xDone,
44   xError:= "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stReadParameter_PSx3xx.stOutput_ReadParameter_PSx3xx.xError,
45   udiErrorID:= "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stReadParameter_PSx3xx.stOutput_ReadParameter_PSx3xx.udiErrorID,
46   diValue:= "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stReadParameter_PSx3xx.stOutput_ReadParameter_PSx3xx.diValue,
47   stDrive:= "GVL_Data_Store_PSx3xx".PSE_1.stDriveData_PSx3xx);

```

Figure 8: Allocation of variables from <GVL_Data_Store_PSx3xx> to FB_MC_ReadParameter_PSx3xx

5.6 FB_MC_WriteParameter_PSx3xx

Parameter values can be written to the drive with this FB.

```

49 //FB_MC_WriteParameter_PSx3xx_DB --> instance of "FB_MC_WriteParameter_PSx3xx"
50 FB_MC_WriteParameter_PSx3xx_DB (xExecute:= "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stWriteParameter_PSx3xx.stInput_WriteParameter_PSx3xx.xExecute,
51 uiParameterNumber:= "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stWriteParameter_PSx3xx.stInput_WriteParameter_PSx3xx.uiParameterNumber,
52 diValue:= "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stWriteParameter_PSx3xx.stInput_WriteParameter_PSx3xx.diValue,
53 xActive:= "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stWriteParameter_PSx3xx.stOutput_WriteParameter_PSx3xx.xActive,
54 xDone:= "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stWriteParameter_PSx3xx.stOutput_WriteParameter_PSx3xx.xDone,
55 xError:= "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stWriteParameter_PSx3xx.stOutput_WriteParameter_PSx3xx.xError,
56 udiErrorID:= "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stWriteParameter_PSx3xx.stOutput_WriteParameter_PSx3xx.udiErrorID,
57 stDrive:= "GVL_Data_Store_PSx3xx".PSE_1.stDriveData_PSx3xx);

```

Figure 9: Allocation of variables from <GVL_Data_Store_PSx3xx> to FB_MC_WriteParameter_PSx3xx

5.7 FB_MC_Parametrization_PSx3xx

With this FB all parameters of the drive can be written at once. This is especially useful for the initialisation. A clear structure was chosen, as all parameters are transferred as a block with the data type <ST_Parameter_PSx3xx>.

```

66 //FB_MC_Parametrization_PSx3xx_DB --> instance of "FB_MC_Parametrization_PSx3xx"
67 FB_MC_Parametrization_PSx3xx_DB (xExecute:= "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stParametrization_PSx3xx.stInput_Parametrization_PSx3xx.xExecute,
68 xDeliveryState:= "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stParametrization_PSx3xx.stInput_Parametrization_PSx3xx.xDeliveryState,
69 stParameter:= "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stParametrization_PSx3xx.stInput_Parametrization_PSx3xx.stParameter,
70 xSaveSettings:= "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stParametrization_PSx3xx.stInput_Parametrization_PSx3xx.xSaveSettings,
71 xActive:= "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stParametrization_PSx3xx.stOutput_Parametrization_PSx3xx.xActive,
72 xDone:= "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stParametrization_PSx3xx.stOutput_Parametrization_PSx3xx.xDone,
73 xError:= "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stParametrization_PSx3xx.stOutput_Parametrization_PSx3xx.xError,
74 udiErrorID:= "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stParametrization_PSx3xx.stOutput_Parametrization_PSx3xx.udiErrorID,
75 uiErrorParameter:= "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stParametrization_PSx3xx.stOutput_Parametrization_PSx3xx.uiErrorParameter,
76 stDrive:= "GVL_Data_Store_PSx3xx".PSE_1.stDriveData_PSx3xx);

```

Figure 10: Allocation of variables from <GVL_Data_Store_PSx3xx> to FB_MC_Parametrization_PSx3xx

The following must be observed when using the FB:

For each parameter, there is a variable <xEnable> and a variable <diValue> (depending on the data type of the parameter).

Example

```

185 "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stParametrization_PSx3xx.stInput_Parametrization_PSx3xx.stParameter.
186 stLoopLength_45.xEnable := TRUE;
187 "GVL_Data_Store_PSx3xx".PSE_1.stFunctionBlocks_PSx3xx.stParametrization_PSx3xx.stInput_Parametrization_PSx3xx.stParameter.
188 stLoopLength_45.diValue := 0;

```

Figure 11: Parameter „loop length“ with the value of 0

- By setting <xEnable> = FALSE, the respective parameter is not written.
- Optionally, a delivery state can be requested before setting individual parameters. To do this, the input <xDeliveryState> must be set to TRUE. This sets the values of all parameters to the delivery state (initially without saving).
- Optionally, the written values can also be saved at the end. To do this, the input <xSaveSettings> must be set to TRUE.
- The order of the write accesses is as follows: <xDeliveryState>, parameter 37, 38, 39, 3...und <xSaveSettings>.
- In the event of a written error, the following parameters are no longer written and the values are not saved if the input <xSaveSettings> is set.

5.8 FB_MC_PosParametrization_PSx3xx

This FB can be used to parameterize the position parameters (parameters that affect the displayed actual position). This is especially useful for the initialisation.

```

79 //fbMC_PosParametrization_PSx3xx_1 --> instance of FB_MC_PosParametrization_PSx3xx (1. PSx3xx)
80 fbMC_PosParametrization_PSx3xx_1(
81   xExecute:= GVL_Data_Store_PSx3xx.PSE_1.stFunctionBlocks_PSx3xx.stPosParametrization_PSx3xx.stInput_PosParametrization_PSx3xx.xExecute,
82   uiDirection:= GVL_Data_Store_PSx3xx.PSE_1.stFunctionBlocks_PSx3xx.stPosParametrization_PSx3xx.stInput_PosParametrization_PSx3xx.uiDirection,
83   uiStepsPerTurn:= GVL_Data_Store_PSx3xx.PSE_1.stFunctionBlocks_PSx3xx.stPosParametrization_PSx3xx.stInput_PosParametrization_PSx3xx.uiStepsPerTurn,
84   diLowerLimit:= GVL_Data_Store_PSx3xx.PSE_1.stFunctionBlocks_PSx3xx.stPosParametrization_PSx3xx.stInput_PosParametrization_PSx3xx.diLowerLimit,
85   diUpperLimit:= GVL_Data_Store_PSx3xx.PSE_1.stFunctionBlocks_PSx3xx.stPosParametrization_PSx3xx.stInput_PosParametrization_PSx3xx.diUpperLimit,
86   diSetPoint:= GVL_Data_Store_PSx3xx.PSE_1.stFunctionBlocks_PSx3xx.stPosParametrization_PSx3xx.stInput_PosParametrization_PSx3xx.diSetPoint,
87   xSaveSettings:= GVL_Data_Store_PSx3xx.PSE_1.stFunctionBlocks_PSx3xx.stPosParametrization_PSx3xx.stInput_PosParametrization_PSx3xx.xSaveSettings,
88   xActive:= GVL_Data_Store_PSx3xx.PSE_1.stFunctionBlocks_PSx3xx.stPosParametrization_PSx3xx.stOutput_PosParametrization_PSx3xx.xActive,
89   xDone:= GVL_Data_Store_PSx3xx.PSE_1.stFunctionBlocks_PSx3xx.stPosParametrization_PSx3xx.stOutput_PosParametrization_PSx3xx.xDone,
90   xError:= GVL_Data_Store_PSx3xx.PSE_1.stFunctionBlocks_PSx3xx.stPosParametrization_PSx3xx.stOutput_PosParametrization_PSx3xx.xError,
91   udiErrorID:= GVL_Data_Store_PSx3xx.PSE_1.stFunctionBlocks_PSx3xx.stPosParametrization_PSx3xx.stOutput_PosParametrization_PSx3xx.udiErrorID,
92   uiErrorParameter:= GVL_Data_Store_PSx3xx.PSE_1.stFunctionBlocks_PSx3xx.stPosParametrization_PSx3xx.stOutput_PosParametrization_PSx3xx.uiErrorParameter,
93   stDrive:= GVL_Data_Store_PSx3xx.PSE_1.stDriveData_PSx3xx);
94

```

Figure 12: Allocation of variables from <GVL_Data_Store_PSx3xx> to FB_MC_PosParametrization_PSx3xx

The following must be observed when using the FB:

- All values must be set and the values must be in a meaningful context to each other. All values are processed, after which the following parameters are written in the specified order:
 1. Direction of rotation (Parameter 37) → <uiDirection>
 2. Position scaling numerator (Parameter 38) → 400
 3. Position scaling denominator (Parameter 39) → <uiStepsPerTurn>
 4. Actual value (Parameter 3) → <diSetPoint>
 5. If (<diSetPoint> > <diUpperLimit>):
 - Upper mapping end (Parameter 41) = <diSetPoint> + (3 x <uiStepsPerTurn>)
 - Else:
 - Upper mapping end (parameter 41) = <diUpperLimit> + (3 x <uiStepsPerTurn>)
 6. Upper limit (parameter 42) → <diUpperLimit>
 7. Lower limit (parameter 43) → <diLowerLimit>

Subsequently the conditions and error codes are listed below, which are displayed, if a condition is not satisfied.

Condition	<udiErrorID>	<uiErrorParameter>
<uiStepsPerTurn> ≥ 1	16#61400	39
<uiStepsPerTurn> ≤ 10000	16#61400	39
<diLowerLimit> ≤ <diUpperLimit>	16#61400	42
(<diUpperLimit> – <diLowerLimit>) / <uiStepsPerTurn> ≤ 250	16#61400	43
If <diSetPoint> < <diLowerLimit>: (<diUpperLimit> – <diSetPoint>) / <uiStepsPerTurn> ≤ 250	16#61400	3
If <diSetPoint> > <diUpperLimit>: (<diSetPoint> – <diLowerLimit>) / <uiStepsPerTurn> ≤ 250	16#61400	3

- Optionally at the end, the written values might be saved permanently. To do this, the input <xSaveSettings> has to be set to TRUE before the execution of the FB.
- In case of an error while writing a parameter, the subsequent parameters are not written anymore. In addition, no saving of the values is carried out, if the input <xSaveSettings> is set.

6 Description of the parameters

6.1 <xActive>

Block	FB_MC_Move_PSx3xx, FB_MC_Parametrization_PSx3xx, FB_MC_PosParametrization_PSx3xx, FB_MC_ReadParameter_PSx3xx, FB_MC_WriteParameter_PSx3xx	
FBs is active or movement order or rather movement is active		
Data type	Bool	
Default	-	
Type	Output	
Description		
<u>FB MC Move PSx3xx</u>		
This output is set when		
<ul style="list-style-type: none">the release for moving <xExecute> is set from FALSE to TRUEthe release for moving <xExecute> is already available and the target position changes, the bit “Drive is running” is set in the status word of the drive (e.g. when readjusting the drive)		
This output is reset when		
<ul style="list-style-type: none">At the end of a run, the bit “Drive is running” is no longer set in the status of the drive and the time of the bus cycle parameter (also see 6.24 <tTimeBusCycle>) has expireda communication error occurs		
<u>All other blocks</u>		
The bit is set as long as the respective function block is running. As soon as the process has been completed or an error has occurred, the bit is reset.		

6.2 <diActualPosition>

Block	FB_MC_Move_PSx3xx	
Actual value		
Data type	DInt	
Default	-	
Type	Output	
Description		
This value is an image of the actual position. If a communication error occurs, the value is set to 0.		

6.3 <xDeliveryState>

Block	FB_MC_Parametrization_PSx3xx
Loading the factory settings (without saving)	
Data type	Bool
Default	FALSE
Type	Input
Description	
However, the station name and IP address remain unaffected.	

6.4 <uiDirection>

Block	FB_MC_PosParametrization_PSx3xx
Direction of rotation	
Data type	UInt
Default	0
Type	Input
Description	
<p>This parameter corresponds to parameter number 37 "direction of rotation".</p> <p>Direction in which the drive should rotate for larger values (when looking at the output shaft):</p> <p>(0 → clockwise; 1 → counterclockwise)</p>	

6.5 <xDone>

Block	FB_MC_Move_PSx3xx, FB_MC_Parametrization_PSx3xx, FB_MC_PosParametrization_PSx3xx, FB_MC_ReadParameter_PSx3xx, FB_MC_WriteParameter_PSx3xx
Target position reached or FBs is done	
Data type	Bool
Default	-
Type	Output
Description	
<p><u>FB MC Move PSx3xx</u></p> <p>This output is an image of the status bit "target position reached". In addition, the output is set to 0 for the duration of the parameter "bus cycle" (also see 6.24 <timeBusCycle>) after the start by <xExecute>. If a communication error occurs, it will be reset.</p> <p><u>All other blocks</u></p> <p>The bit is set as soon as the process has been successfully completed. It is reset at the start of the respective process.</p>	

6.6 <stDrive>

Block	FB_MC_Error_PSx3xx, FB_MC_Move_PSx3xx, FB_MC_Parametrization_PSx3xx, FB_MC_PosParametrization_PSx3xx, FB_MC_ReadParameter_PSx3xx, FB_MC_WriteParameter_PSx3xx
Data structure for communication	
Data type	ST_DriveData_PSx3xx
Type	Input & Output
Description	
A global instance of this structure is required for each drive. This instance is transferred to each function block (FB) that acts on the operated drive.	

6.7 <xError>

Block	FB_MC_Error_PSx3xx, FB_MC_Move_PSx3xx, FB_MC_Parametrization_PSx3xx, FB_MC_PosParametrization_PSx3xx, FB_MC_ReadParameter_PSx3xx, FB_MC_WriteParameter_PSx3xx
FB execution error or drive error	
Data type	Bool
Default	FALSE
Type	Output
Description	
<p><u>FB MC Move PSx3xx</u> The error bit is updated every cycle and is set, if an error occurs during execution of the FB. It can also be set, while the drive is active (e.g. drag error).</p> <p><u>FB MC Error PSx3xx</u> The output <xError> is updated every cycle as long as <xExecute> is set. If <xExecute> is reset, this output assumes the specified default value.</p> <p><u>All other blocks</u> The error bit is updated every cycle and is set, if an error occurred during an execution of the FB.</p>	

6.8 <sErrorDescription>

Block	FB_MC_Error_PSx3xx
Error description as text output	
Data type	String[254]
Default	""
Type	Output
Description	
An error description as text output	

6.9 <udiErrorID>

Block	FB_MC_Error_PSx3xx, FB_MC_Move_PSx3xx, FB_MC_Parametrization_PSx3xx, FB_MC_PosParametrization_PSx3xx, FB_MC_ReadParameter_PSx3xx, FB_MC_WriteParameter_PSx3xx
Error identifier (also see chapter 4 Error description (<udiErrorID>))	
Data type	UDint
Default	0
Type	Output
Description	
The error identifier can also be set, while the drive is moving. (e.g. drag error)	
<u>FB MC Move PSx3xx</u>	
The error identifier is updated every cycle and is set, if an error occurs during execution of the FB. It can also be set, while the drive is active (e.g. drag error).	
<u>FB MC Error PSx3xx</u>	
The output <udiErrorID> is updated every cycle as long as <xExecute> is set. If <xExecute> is reset, this output assumes the specified default value.	
<u>All other blocks</u>	
The error identifier bit is updated every cycle and is set, if an error occurred during execution of the FB.	



CAUTION!

The outputs <xError> and <udiErrorID> of the function blocks are always updated - even if the <xExecute> input is not set, except for **MC_Error_PSx3xx**.

6.10 <uiErrorParameter>

Block	FB_MC_Parametrization_PSx3xx, FB_MC_PosParametrization_PSx3xx
Parameter number at which the error occurred	
Data type	UInt
Default	0
Type	Output
Description	
If there was no error, the value 0 is returned.	

6.11 <xExecute>

Block	FB_MC_Error_PSx3xx, FB_MC_Move_PSx3xx, FB_MC_Parametrization_PSx3xx, FB_MC_PosParametrization_PSx3xx, FB_MC_ReadParameter_PSx3xx, FB_MC_WriteParameter_PSx3xx	
Release of the drive		
Data type	Bool	
Default	FALSE	
Type	Input	
Description		
<u>FB MC Move PSx3xx</u> A setpoint is only approached when this input is set. This input acts directly on the enable bit (bit 4) in the control word. If the input remains set and the readjustment is active in the drive, the drive readjusts automatically. If the input is set and the setpoint is changed, the drive moves to it immediately. An edge is not required. If the input is reset during a movement, the drive stops.		
<u>All other blocks</u> With a rising edge, the process of the respective function block is executed. A rising edge must be generated again for a new process. If the bit is reset, the outputs assume the specified default values.		

6.12 <uiInputAddress>

Block	FC_MC_Communication_PSx3xx_...	
Address of the input module of the drive		
Data type for S7-1200 S7-1500	UInt	
Data type for S7-300	DWord	
Type	Input	
Description		
A detailed description of the address assignment can be found in the chapter 2.1 <ST_DriveData_PSx3xx>		

6.13 <diLowerLimit>

Block	FB_MC_PosParametrization_PSx3xx
Lower limit	
Data type	DInt
Default	0
Type	Input
Description	
This parameter corresponds to parameter 43 "lower limit".	

6.14 <xManualRunToLargerValues>

Block	FB_MC_Move_PSx3xx
Manual run to larger values	
Data type	Bool
Default	FALSE
Type	Input
Description	
Manual run is used to increase the values up to the upper limit. The input <xExecute> must also be set.	



CAUTION!

When resetting the input <xManualRunToLargerValues> <xExecute> must also be reset, otherwise the drive will approach the target position <diTargetPosition>.

6.15 <xManualRunToSmallerValues>

Block	FB_MC_Move_PSx3xx
Manual run to smaller values	
Data type	Bool
Default	FALSE
Type	Input
Description	
Manual run is used to decrease the values down to the lower limit. The input <xExecute> must also be set.	



CAUTION!

When resetting the input <xManualRunToSmallerValues>, <xExecute> must also be reset, otherwise the drive will approach the target position <diTargetPosition >.

6.16 <uiOutputAddress>

Block	FC_MC_Communication_PSx3xx_...
Address of the output module of the drive	
Data type for S7-1200 S7-1500	UInt
Data type for S7-300	DWord
Type	Input
Description	
A detailed description of the address assignment can be found in the chapter 2.1 <ST_DriveData_PSx3xx>.	

6.17 <stParameter>

Block	FB_MC_Parametrization_PSx3xx
Transfer of the parameter set	
Data type	ST_Parameter_PSx3xx
Default	-
Type	Input
Description	
The parameter number is specified after the parameter name. The data type, a description and the range of values can be found in the instruction manual PSx-3xx-EC.	

6.18 <uiParameterNumber>

Block	FB_MC_WriteParameter_PSx3xx, FB_MC_ReadParameter_PSx3xx
Parameter number of the parameter to be read or written	
Data type	UInt
Default	0
Type	Input
Description	
A read or write process is started with a rising edge. A rising edge must be generated again for a new process. If the bit is reset, the outputs assume the specified initial values.	

6.19 <xSaveSettings>

Block	FB_MC_Parametrization_PSx3xx, FB_MC_PosParametrization_PSx3xx
Saving the parameter settings	
Data type	Bool
Default	FALSE
Type	Input
Description	
This parameter corresponds to parameter 96 „delivery state. (function <Writing “1”>)	

6.20 <diSetPoint>

Block	FB_MC_PosParametrization_PSx3xx
Setpoint of the measuring system	
Data type	DInt
Default	0
Type	Input
Description	
This parameter corresponds to parameter 3 “actual value”.	

6.21 <uiStepsPerTurn>

Block	FB_MC_PosParametrization_PSx3xx
Steps per rotation on the output shaft (resolution)	
Data type	Int
Default	0
Type	Input
Description	
The number of steps per rotation <uiStepsPerTurn> directly results in the value of the parameter “position scaling denominator” (Parameter 39). It is assumed that the value of “position scaling nominator” (Parameter 38) is in the delivery state (400).	

6.22 <usiSubIndex>

Block	FB_MC_ReadParameter_PSx3xx, FB_MC_WriteParameter_PSx3xx
Subindex of the parameters	
Data type	USInt
Default	0
Type	Input
Description	
The value of the parameter <usiSubIndex> can be left at the default value. Other values are currently not planned.	

6.23 <diTargetPosition>

Block	FB_MC_Move_PSx3xx
Target position to be approached	
Data type	DInt
Default	0
Type	Input
Description	
If a new target position is set during a run, it is approached immediately. If <xExecute> is still set after the end of the run and the target position is changed, the drive moves to it immediately.	



INFORMATION!

In order to approach the same target position, e.g. after a blocking error, the release of the drive <xExecute> must be reset and set again.

6.24 <tTimeBusCycle>

Block	FB_MC_Move_PSx3xx
Cycle time of the Profinet bus cycle	
Data type	Time
Default	40ms
Type	Input
Description	
<p>With long cycle times on the Profinet-Bus, it can happen that the PLC does not receive a change in the bit "drive running". This is the case when the time for the run command is shorter than the bus cycle.</p> <p>To counteract this, the parameter <tTimeBusCycle> has been implemented. 40 ms are set as the initial value for a bus cycle. For this duration, the output <xActive> is always set and the output <xDone> is reset after a movement order. After that, these outputs assume the corresponding state depending on the feedback of the status word (bit "drive running" and bit "target position reached").</p> <p>In case of longer cycle times of the bus cycle, it is advisable to set the actual duration of the cycle multiplied by 2 instead of the initial value.</p> <p>If a shorter bus cycle time can be kept for sure, the time for positioning is very short and the higher-level sequence is to be continued quickly after positioning is complete, the value for <tTimeBusCycle> can also be reduced.</p>	

6.25 <diUpperLimit>

Block	FB_MC_PosParametrization_PSx3xx	
Upper limit		
Data type	DInt	
Default	0	
Type	Input	
Description		
This parameter corresponds to parameter 42 “upper limit”.		

6.26 <diValue>

Block	FB_MC_ReadParameter_PSx3xx, FB_MC_WriteParameter_PSx3xx	
value to be written or read value of a parameter		
Data type	DInt	
Default	0	
Type	Input for FB_MC_WriteParameter_PSx3xx Output for FB_MC_ReadParameter_PSx3xx	
Description		
A read or write process is started with a rising edge. A rising edge must be generated again for a new process.		

6.27 <sValue>

Block	FB_MC_ReadParameter_PSx3xx	
Value of a parameter to be read		
Data type	String[31]	
Default	0	
Type	Output	
Description		
A reading process is started with a rising edge. A rising edge must be generated again for a new process.		

7 Application of the function blocks

The function blocks are programmed in TIA V13. Upgrades to higher versions can be done without any problems.

There are two ways of using the function blocks:

- As Project or
- As embedded library

7.1 Project

Opening and upgrading the project depending on the CPU:

- example_hw_PSx3xx_1200_1500_xxxxxxx.zap13
→ for SIMATIC S7 - 1200 and SIMATIC S7 – 1500
- example_hw_PSx3xx_300_xxxxxxx.zap13
→ for SIMATIC S7 - 300

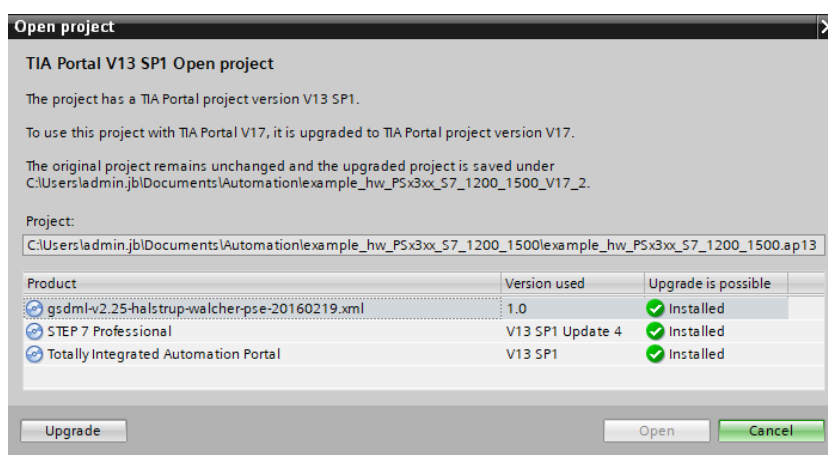


Figure 13: Upgrade of a project in TIA V13 to TIA V17

7.2 Library

Opening and upgrading the library depending on the CPU:

- library_PSx3xx_1200_1500_xxxxxxx.zal13
→ for SIMATIC S7 - 1200 and SIMATIC S7 – 1500
- library_PSx3xx_300_xxxxxxx.zal13
→ for SIMATIC S7 - 300

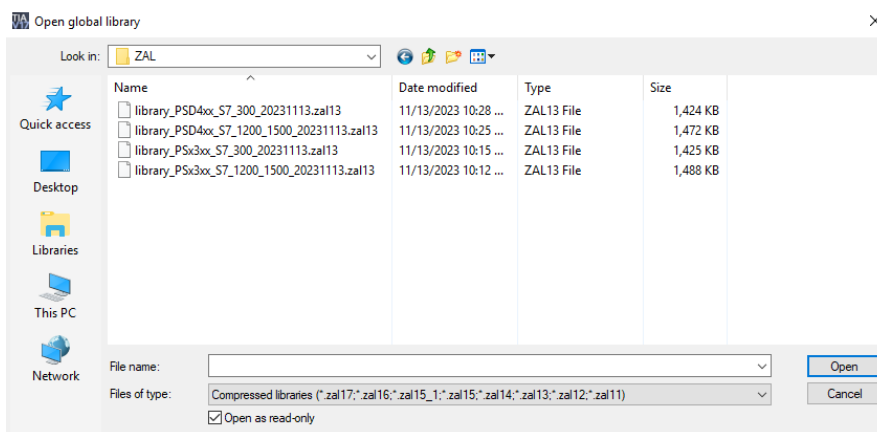


Figure 14: Opening a compressed library

After opening and upgrading the library, the function blocks appear as below:

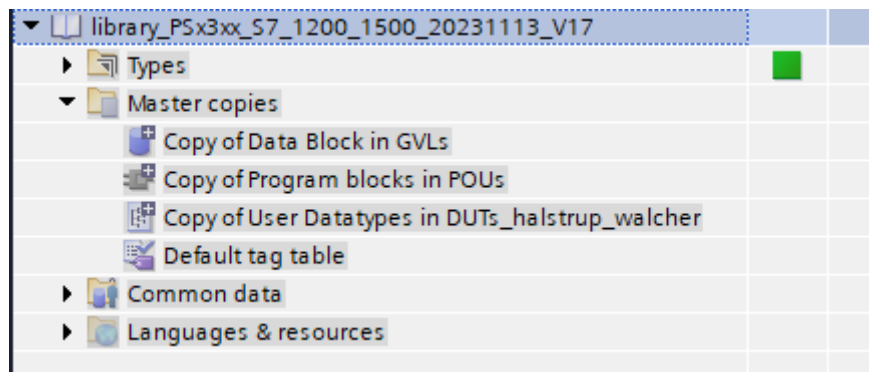


Figure 15: Global library in TIA V17

The following function from the library must always be copied into the user's project (irrespective of the **FB_MC_..._PSx3xx** function blocks actually used and the number of drives):

- **FB_MC_Error_ID_PSx3xx**

This function is used for error detection with the drive.

In addition, the required function blocks **FB_MC_..._PSx3xx** must be copied to the user project. All or a selection of the following blocks can be used:

- **FB_MC_Move_PSx3xx**
- **FB_MC_Error_PSx3xx**
- **FB_MC_ReadParameter_PSx3xx**
- **FB_MC_WriteParameter_PSx3xx**
- **FB_MC_Parametrization_PSx3xx**
- **FB_MC_PosParametrization_PSx3xx**

Additionally, the global variable list <GVL_Data_Store_PSx3xx> can be adopted as a template for the connection to the function blocks. However, this requires all data types (DUTs_halstrup_walcher)

For the inclusion of the sample programs the global variable list <GVL_VariablesForProgram> should be adopted.

7.3 Interlock between the function blocks

The blocks are partially locked against each other. This ensures that not two accesses out of different function blocks can be executed at the same time.

Es gelten die folgenden Regeln:

There are following rules:

- If the input <xExecute> of **FB_MC_Move_PSx3xx** is set, the function blocks **FB_MC_Parametrization_PSx3xx** und **FB_MC_PosParametrization_PSx3xx** cannot be activated (<udiErrorID>: 16#x1000).
- On the other hand, it is possible to activate **FB_MC_ReadParameter_PSx3xx** or **FB_MC_WriteParameter_PSx3xx** during a movement (e.g. to read out the current torque or to change the target speed during the actual movement).
- If the input <xExecute> of **FB_MC_Parametrization_PSx3xx** or **FB_MC_PosParametrization_PSx3xx** is set, the function block **FB_MC_Move_PSx3xx** cannot be activated (<udiErrorID>: 16#01000).

- Only one of the function blocks **FB_MC_ReadParameter_PSx3xx**, **FB_MC_WriteParameter_PSx3xx**, **FB_MC_Parametrization_PSx3xx** or **FB_MC_PosParametrization_PSx3xx** can be active at the same time. If the input <xExecute> of a function block is set and the input <xExecute> of another function block is set, then there is the <udiErrorID> 16#x1000.
- The function block **FB_MC_Error_PSx3xx** can always be activated.

7.4 Several PSx3xx in a project

There is only one PSx3xx in the example projects. If several drives are used, the project must be adjusted accordingly.

For example for three PSx3xx:

1. Initialize instance data blocks
 - Three instance data blocks of fb's **FB_MC_Move_PSx3xx**, e.g.
 - fbMC_Move_PSx3xx_1
 - fbMC_Move_PSx3xx_2
 - fbMC_Move_PSx3xx_3
 - Three instance data blocks of fb's **FB_MC_Error_PSx3xx**, e.g.
 - fbMC_Error_PSx3xx_1
 - fbMC_Error_PSx3xx_2
 - fbMC_Error_PSx3xx_3
 - further required instance data blocks
2. Afterwards, the global variable list <GVL_Data_Store_PSx3xx> is adapted. Three variables of the <ST_Variables_PSx3xx> type are created there, e.g. with the following designations:
 - „PSE_1“
 - „PSE_2“
 - „PSE_3“
3. Finally, the variables from <GVL_Data_Store_PSx3xx> must be assigned to the function blocks (see chapter 5 Description of the function blocks)

8 Example programs

Example programs are programmed to demonstrate, how the function blocks are to be used. The functions are programmed in SCL and the respective function must be inserted into the main program in order to be able to call the function.

```

15 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
16
17 (* MAIN *)
18 |
19 //call of function
20 "FC_Example_7_Error_Upper_Position_Limit_Exceeded"();
21
22
23 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
24

```

Figure 16: Calling function in the main program

Similarity of all example programs

- for all programs, the required instances are defined and called at the beginning.
- For all programs the variable <xStartProgram> must be set to TRUE for starting the program.
- For each sample program, the first step after starting the program is to set all relevant variables to FALSE for initialization.
- In each example program, the first step after starting the program is to set all relevant variables to FALSE for the initialization.
- The program can be ended with <xStopProgram>.

8.1 Example 1: positioning mode

The positioning mode of the **FB_MC_Move_PSx3xx** is presented in this example program. In this example, the drive moves an endless loop between the values 20000 and 10000.

Sequence of the program

Case	Description
0	All relevant variables are set to FALSE for initialization and the state machine is incremented by one. (It is always useful to set all relevant variables to the initial value at the beginning of a program).
1	The target position of 20000 and the movement command are set. If the actual value is at 20000 ± 2 (see parameter 44 "Positioning window") and the travel was successfully completed, the movement command is reset and the state machine is increased by one.
2	The target position of 10000 and the movement command are set. If the actual value is at 10000 ± 2 (see parameter 44 "Positioning window") and the travel has been carried out successfully, the movement command is reset and the State Machine is set to 1.

8.2 Example 2: manual mode

This sample program presents the manual mode of the **FB_MC_Move_PSx3xx**. In this example, the drive moves to a start position with the positioning mode and then with the manual mode.

Sequence of the program

Case	Description
0	All relevant variables are set to FALSE for initialization and the state machine is incremented by one. (It is always useful to set all relevant variables to the initial value at the beginning of a program).
1	The start position of 20000 and the movement command are set. If the current value is at 20000 ± 2 (see parameter 44 "Positioning window") and the movement was successfully completed, then the movement command is reset and the State Machine is increased by one.
2	With manual mode, travel is performed over the target of 30000. Only when the actual value corresponds to 30000, the movement command for manual move is reset and the state machine is set to 100.

8.3 Example 3: read actual value

This sample program introduces the **FB_MC_ReadParameter_PSx3xx**.

In this example, the drive moves to a target position and the parameter "actual value" is read.

Sequence of the program

Case	Description
0	All relevant variables are set to FALSE for initialization and the state machine is incremented by one. (It is always useful to set all relevant variables to the initial value at the beginning of a program).
1	The start position of 20000 and the movement command are set. If the current value is at 20000 ± 2 (see parameter 44 "positioning window") and the movement was successfully completed, then the movement command is reset and the State Machine is increased by one.
2	The parameter "actual value" is read. If the value is 20000 ± 2 (see parameter 44 "Positioning window"), then the read command is reset and the state machine is set to 100.

8.4 Example 4: writing the delivery state

This sample program introduces the **FB_MC_WriteParameter_PSx3xx**. The "Delivery state" parameter is written in this example and the drive then moves to its range center.

Sequence of the program

Case	Description
0	All relevant variables are set to FALSE for initialization and the state machine is incremented by one. (It is always useful to set all relevant variables to the initial value at the beginning of a program).
1	The start position of 20000 and the movement command are set. If the current value is at 20000 ± 2 (see parameter 44 "positioning window") and the travel was successfully completed, then the movement command is reset and the State Machine is increased by one.
2	The value of the "Delivery state" parameter is written to -5. The drive is reset to its delivery state, the parameters are set to the default value and additionally a positioning to the measuring range center (position 51200) takes place. If the write command was executed successfully, the write command is reset and the state machine is increased by one. (The drive still moves to the center of the measuring range).

8.5 Example 5: Parameterize parameters

This sample program presents the **FB_MC_Parametrization_PSx3xx**. The parameters "loop length" and "target speed" are parameterized in this example and then a positioning run is performed.

Sequence of the program

Case	Description
0	All relevant variables are set to FALSE for initialization and the state machine is incremented by one. (It is always useful to set all relevant variables to the initial value at the beginning of a program).
1	The start position of 20000 and the movement command are set. If the current value is at 20000 ± 2 (see parameter 44 "Positioning window") and the travel was successfully completed, then the movement command is reset and the State Machine is increased by one.
2	The "Loop length" and "target speed" parameters are parameterized. The <xEnable> of the parameters must be set. If the parameterization command was executed successfully, the parameterization command is reset and the state machine is increased by one.
3	The target position of 10000 and the movement command are set. During this positioning travel, no loop travel is performed and travel is performed at 20 rpm. If the current value is at 10000 ± 2 (see parameter 44 "Positioning window") and the travel was successfully completed, then the movement command is reset and the state machine is set to 100.

8.6 Example 6: Parameterize position parameters

In this sample program the **FB_MC_PosParametrization_PSx3xx** is presented. Parameters are set and the "current actual position" parameter is read.

Sequence of the program

Case	Description
0	All relevant variables are set to FALSE for initialization and the state machine is incremented by one. (It is always useful to set all relevant variables to the initial value at the beginning of a program).
1	The start position of 20000 and the movement command are set. If the current value is at 20000 ± 2 (see parameter 44 "Positioning window") and the travel was successfully completed, then the movement command is reset and the State Machine is increased by one.
2	All position parameters are parameterized. The parameter "Actual position" is set to 0, the parameter "Upper end limit" is set to 10000 and the parameter "Lower end limit" is set to -10000. The rest of the parameters are not changed and the parameter values are not to be saved. If the parameterization command of the position parameters was executed successfully, then the parameterization command of the position parameters is reset and the State Machine is increased by one.
3	The parameter "current position" is read. If the value is 0 ± 2 , then the read command is reset and the state machine is set to 100.

8.7 Example 7: Error upper end limit reached

In this sample program the **FB_MC_Error_PSx3xx** is presented. It is moved to the upper end limit and the error message is read out.

Sequence of the program

Case	Description
0	All relevant variables are set to FALSE for initialization and the state machine is incremented by one. (It is always useful to set all relevant variables to the initial value at the beginning of a program).
1	The start position of 20000 and the movement command are set. If the current value is at 20000 ± 2 (see parameter 44 "Positioning window") and the travel was successfully completed, then the movement command is reset and the State Machine is increased by one.
2	The parameter "Upper end limit" is written to the value 25000. If the write command was executed successfully, then the write command is reset and the state machine is incremented by one.
3	The function block FB_MC_Error_PSx3xx is activated to detect errors.
4	With manual mode the upper end limit is approached. Only when the actual value corresponds to 25000, the error description is displayed as a string. Then the move command for manual mode is reset and the index is set to 100. (It would also be possible to receive this error from FB_MC_Move_PSx3xx (<udiErrorID>). The <udiErrorID> would then be 0x100B0.)

List of Figures

Figure 1: Addresses of the process data from the perspective of the PLC.....	6
Figure 2: Addresses to be set from the perspective of the device.....	7
Figure 3: Allocation of the variables from <GVL_Data_Store_PSx3xx> to FB_MC_Move_PSx3xx	8
Figure 4: <GVL_Data_Store_PSx3xx> with the variables for a drive	8
Figure 5: Zuweisung der Variablen von <GVL_Data_Store_PSx3xx> zu der FC FC_MC_Communication_PSx3xx_1200_1500	11
Figure 6: Allocation of variables from DB <Data_Store> to FB_MC_Move_PSx3xx	11
Figure 7: Allocation of variables from DB <GVL_Data_Store_PSx3xx> to FB_MC_Error_PSx3xx	12
Figure 8: Allocation of variables from <GVL_Data_Store_PSx3xx> to FB_MC_ReadParameter_PSx3xx	12
Figure 9: Allocation of variables from <GVL_Data_Store_PSx3xx> to FB_MC_WriteParameter_PSx3xx	13
Figure 10: Allocation of variables from <GVL_Data_Store_PSx3xx> to FB_MC_Parametrization_PSx3xx	13
Figure 11: Parameter „loop length“ with the value of 0	13
Figure 12: Allocation of variables from <GVL_Data_Store_PSx3xx> to FB_MC_PosParametrization_PSx3xx	14
Figure 13: Upgrade of a project in TIA V13 to TIA V17	25
Figure 14: Opening a compressed library.....	25
Figure 15: Global library in TIA V17	26
Figure 16: Calling function in the main program.....	28