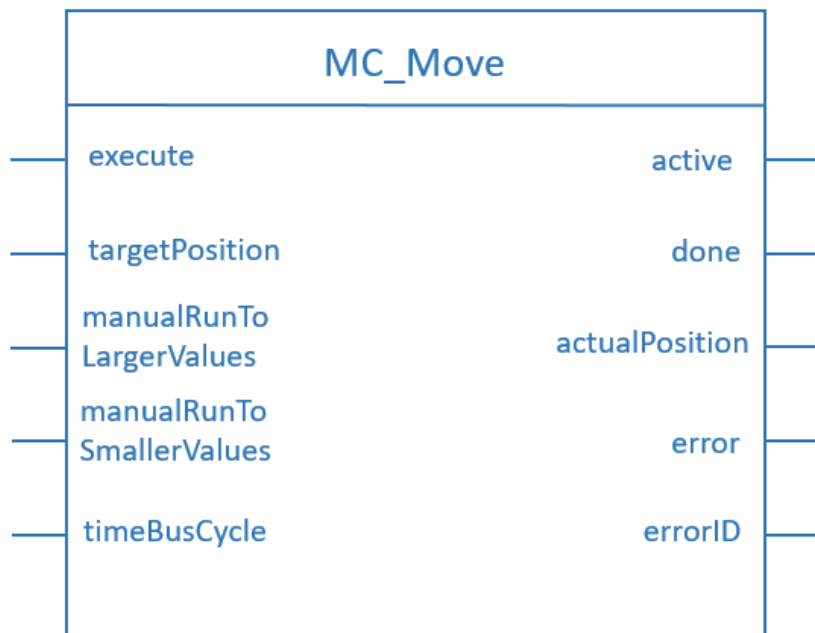


## Function Blocks PSD4xx for TIA-Portal

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



## Purpose of instruction manual

This instruction manual describes the function blocks for the PSD4xx 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](mailto:info@halstrup-walcher.de)  
[www.halstrup-walcher.de](http://www.halstrup-walcher.de)

© 2023

**11.04.2023, TS & JB**

**7100.006644B version 2.0 Instruction Manual Function Blocks PSD4xx PN**

## Table of Contents

1	Safety precautions.....	5
1.1	Appropriate use.....	5
1.2	Symbols.....	5
2	PLC data types.....	6
2.1	driveData_PSD4xx.....	6
2.2	Int_type_PSD4xx.....	7
2.3	DInt_Type_PSD4xx.....	7
2.4	parameter_PSD4xx.....	8
3	Data block Data_Store (DB100).....	9
4	Error description (<errorID>).....	10
5	Description of the function blocks.....	12
5.1	MC_Communication_PSD4xx_S7-1200_1500 (FC101), MC_Communication_PSD4xx_S7-300 (FC101).....	12
5.2	MC_Move_PSD4xx (FB110).....	12
5.3	MC_Error_PSD4xx (FB111).....	13
5.4	MC_Error_ID_PSD4xx (FC100).....	13
5.5	MC_ReadParameter_PSD4xx (FB112).....	14
5.6	MC_WriteParameter_PSD4xx (FB113).....	14
5.7	MC_Parametrization_PSD4xx (FB114).....	14
5.8	MC_PosParametrization_PSD4xx (FB115).....	15
6	Description of the parameters.....	17
6.1	<active>.....	17
6.2	<actualPosition>.....	17
6.3	<communicationError>.....	18
6.4	<deliveryState>.....	18
6.5	<direction>.....	18
6.6	<done>.....	19
6.7	<drive>.....	19
6.8	<error>.....	20
6.9	<errorDescription>.....	20
6.10	<errorID>.....	21
6.11	<errorParameter>.....	21
6.12	<execute>.....	22
6.13	<inputAddress>.....	22
6.14	<lowerLimit>.....	22
6.15	<manuelRunToLargerValues>.....	23

6.16	<manualRunToSmallerValues> .....	23
6.17	<outputAddress> .....	23
6.18	<parameter> .....	24
6.19	<parameterNumber> .....	24
6.20	<saveSettings> .....	24
6.21	<setPoint> .....	25
6.22	<stepsPerTurn> .....	25
6.23	<subIndex> .....	25
6.24	<targetPosition> .....	26
6.25	<timeBusCycle> .....	26
6.26	<upperLimit> .....	27
6.27	<value> .....	27
7	Application of the function blocks .....	28
7.1	Project .....	28
7.2	Library .....	28
7.3	Interlock between the function blocks .....	30
7.4	Several PSD4xx in a project .....	30
8	Example programs .....	32
8.1	Example 1: positioning mode .....	32
8.2	Example 2: manual mode .....	33
8.3	Example 3: read actual value .....	33
8.4	Example 4: write direction of rotation .....	34
8.5	Example 5: parameterize parameters .....	35
8.6	Example 6: parameterize position parameters .....	35
8.7	Example 7: error upper limit exceeded .....	36
	List of Figures .....	37

## 1 Safety precautions

### 1.1 Appropriate use

The positioning systems PSD4xx 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.

**PSD4xx 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 PLC data types

### 2.1 driveData\_PSD4xx

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
<pdAddressIn>	Int (S7-1200 and S7-1500) DWord (S7-300)	User	Address process data (Device → Controller)
<pdAddressOut>	Int (S7-1200 and S7-1500) DWord (S7-300)	User	Address process data (Controller → Device)
<axisName>	String[16]	User (optional)	Name of axis
<axisDescription>	String[32]	User (optional)	Description (e. g. function, task of this axis)
<state>	DInt	Function blocks	Actual state
<communicationError>	Bool	Function blocks	Communication error to IO-Device
<private>	Struct	Function blocks	Data structure for internal use

The following diagram shows how the given process data addresses may be checked in the TIA Portal:

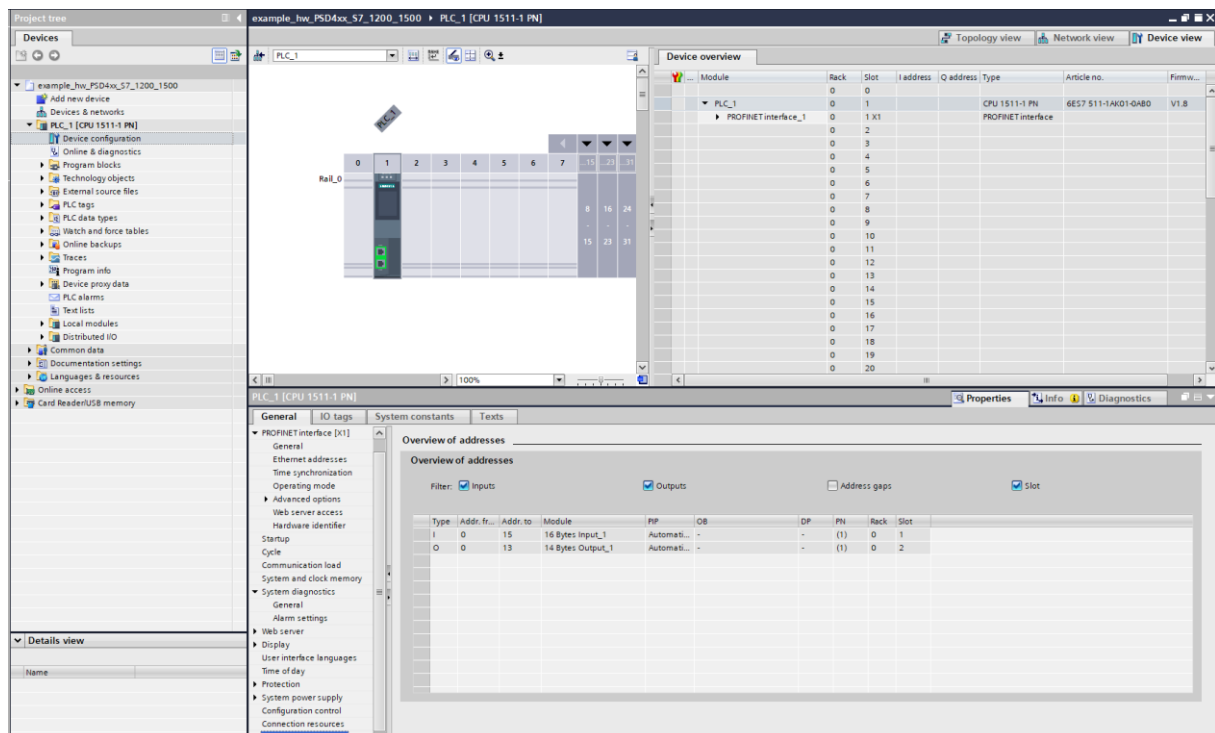


Figure 1: Addresses of the process data from the perspective of the PLC

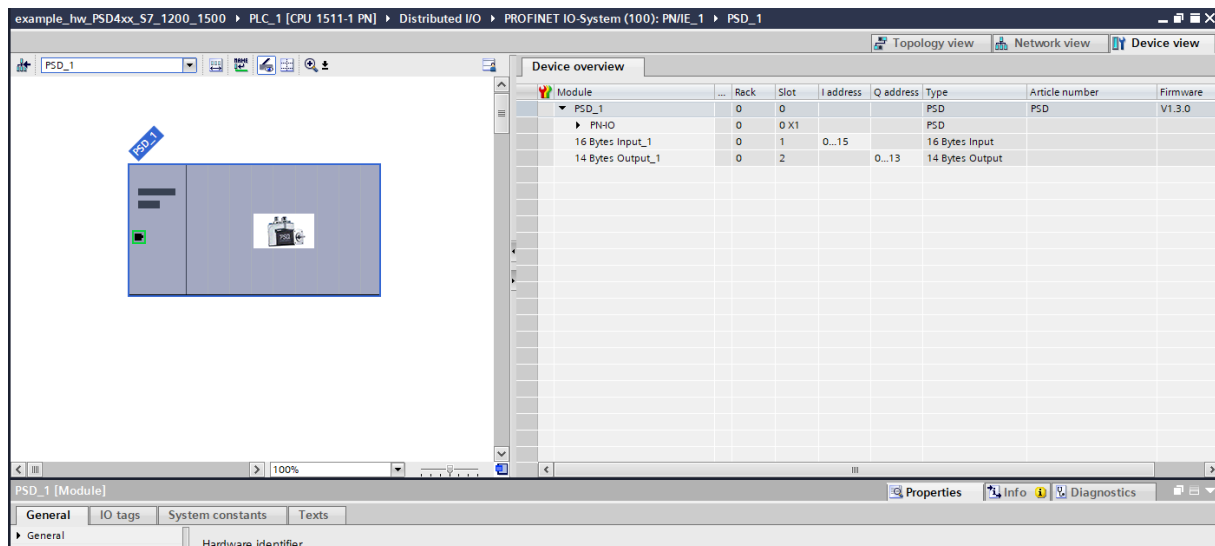


Figure 2: Addresses to be set from the perspective of the device

## 2.2 Int\_type\_PSD4xx

example\_hw\_PSD4xx\_S7\_1200\_1500 ▶ PLC\_1 [CPU 1511-1 PN] ▶ PLC data types ▶ Int\_type\_PSD4xx

	Name	Data type	Default value	Accessible f...	Visible in ...	Setpoint	Comment
1	enable	Bool	false	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	enable
2	value	Int	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	value

Figure 3: Data type <Int\_type\_PSD4xx>

**This data type is only required together with the FB MC\_Parametrization\_PSD4xx.**

The data type is contained as a substructure in the data type <parameter\_PSD4xx>. It is used for all parameters of type <Integer> (16 bits).

## 2.3 DInt\_Type\_PSD4xx

example\_hw\_PSD4xx\_S7\_1200\_1500 ▶ PLC\_1 [CPU 1511-1 PN] ▶ PLC data types ▶ DInt\_type\_PSD4xx

	Name	Data type	Default value	Accessible f...	Visible in ...	Setpoint	Comment
1	enable	Bool	false	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	enable
2	value	DInt	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	value

Figure 4: Data type <DInt\_type\_PSD4xx>

**This data type is only required together with the FB MC\_Parametrization\_PSD4xx.**

The data type is contained as a substructure in the data type <parameter\_PSD4xx>. It is used for all parameters of type <Double Integer> (32 bits).

## 2.4 parameter\_PSD4xx

example\_hw\_PSD4xx\_S7\_1200\_1500 ▶ PLC\_1 [CPU 1511-1 PN] ▶ PLC data types ▶ parameter\_PSD4xx

parameter_PSD4xx							
	Name	Data type	Default value	Accessible f...	Visible in ...	Setpoint	Comment
1	actualValue_3	"DInt_type_PSD..."		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	actual value
2	enable	Bool	false	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	enable
3	value	DInt	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	value
4	dirRotation_37	"Int_type_PSD4xx"		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	direction of rotation
5	enable	Bool	false	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	enable
6	value	Int	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	value
7	posScaleNumerator_38	"Int_type_PSD4xx"		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	position scaling numerator
8	enable	Bool	false	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	enable
9	value	Int	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	value
10	posScaleDenominator_39	"Int_type_PSD4xx"		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	position scaling denominator
11	referencingValue_40	"DInt_type_PSD4xx"		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	referencing value
12	upperMappingEnd_41	"DInt_type_PSD4xx"		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	upper mapping end
13	upperLimit_42	"DInt_type_PSD4xx"		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	upper limit
14	lowerLimit_43	"DInt_type_PSD4xx"		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	lower limit
15	positioningWindow_44	"Int_type_PSD4xx"		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	positioning window
16	loopLength_45	"DInt_type_PSD4xx"		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	length of loop

Figure 5: Extract of data type <parameter\_PSD4xx>

**This data type is only required together with the FB MC\_Parametrization\_PSD4xx.**

It serves as a template for setting the individual parameters and is contained in the data block <Data\_Store>.

With <enable> set to TRUE, the respective parameter is enabled for writing.

<value> contains the value to be written.



### INFORMATION!

Don't forget to set the <enable> back to FALSE after setting the <execute> to FALSE!



### 3 Data block Data\_Store (DB100)

This data block serves as a template for the assignment to the function blocks. The DB provides the necessary variables to be able to set all inputs and outputs of all available function blocks.

Each variable currently present only once. If there are several PSD4xx in a project, the number of variables must be copied.

#### Example

```

26 // "MC_Move_PSD4xx_DB" --> instance of "MC_Move_PSD4xx"
27 "MC_Move_PSD4xx_DB" (execute := "Data_Store".PSE_Move_1.input.execute,
28     targetPosition := "Data_Store".PSE_Move_1.input.targetPosition,
29     manualRunToLargerValues := "Data_Store".PSE_Move_1.input.manualRunToLargerValues,
30     manualRunToSmallerValues := "Data_Store".PSE_Move_1.input.manualRunToSmallerValues,
31     timeBusCycle := "Data_Store".PSE_Move_1.input.timeBusCycle,
32     active => "Data_Store".PSE_Move_1.output.active,
33     done => "Data_Store".PSE_Move_1.output.done,
34     actualPosition => "Data_Store".PSE_Move_1.output.actualPosition,
35     error => "Data_Store".PSE_Move_1.output.error,
36     errorID => "Data_Store".PSE_Move_1.output.errorID,
37     drive := "Data_Store".PSE_1);
--

```

Figure 6: Assignment of variables from DB <Data\_Store> to FB **MC\_Move\_PSD4xx**

If the DB is transferred to the project, we recommend deleting the variables that are not used (so that memory in the PLC is not unnecessarily occupied) and adapting the block to the number of drives that are actually present.

example\_hw\_PSD4xx\_S7\_1200\_1500 ▶ PLC\_1 [CPU 1511-1 PN] ▶ Program blocks ▶ Data\_Store\_PSD4xx [DB100]

<

Figure 7: DB <Data\_Store> with the variables for one device

## 4 Error description (<errorID>)

The error codes that are outputs by the function blocks are described below:

<errorID> (hex)	Description
<b>16#F000 (mask)</b>	<b>Function blocks</b>
16#0xxx	No error code
16#1xxx	Error in MC_Move_PSD4xx
16#2xxx	Error in MC_Error_PSD4xx
16#3xxx	Error in MC_ReadParameter_PSD4xx
16#4xxx	Error in MC_WriteParameter_PSD4xx
16#5xxx	Error in MC_Parametrization_PSD4xx
16#6xxx	Error in MC_PosParametrization_PSD4xx
<b>16#0F00 (mask)</b>	<b>Internal errors in the function blocks and process data errors</b>
16#x0xx	No internal errors in the function blocks and process data errors
16#x1xx	Error in the state machine (interlock)
16#x2xx	Invalid process data input address
16#x3xx	Invalid process data output address
16#x4xx	Error while reading process data
16#x5xx	Error while writing process data
16#x6xx	Forbidden input data change
<b>16#00F0 (mask)</b>	<b>Parameter errors</b>
16#xx0x	No parameter errors
16#xx1x	Parameter: communication timeout (1000 ms)
16#xx2x	Parameter: invalid parameter number
16#xx3x	Parameter: parameter value is read only
16#xx4x	Parameter: below lower limit or upper limit exceeded
16#xx5x	Parameter: invalid subindex
16#xx6x	Parameter: no array
16#xx7x	Parameter: incorrect data type
16#xx8x	Parameter: setting not allowed (resetting only)
16#xx9x	Parameter: request cannot be processed due to operating state
16#xxAx	Other error
<b>16#000F (mask)</b>	<b>Drive errors</b>
16#xxx0	No drive errors
16#xxx1	Drag error
16#xxx2	Under- or overvoltage motor supply (STO-enabling inactive*)
16#xxx3	Positioning run aborted
16#xxx4	Temperature exceeded
16#xxx5	Measuring system error (Measuring system error or STO hardware error*)
16#xxx6	Positioning error (block)
16#xxx7	Manual displacement
16#xxx8	Incorrect target value
16#xxx9	Under- or overvoltage during run / failure voltage control
16#xxxA	Below lower limit
16#xxxB	Upper limit exceeded

\* If the PSD4xx is a device with STO, then the error description in brackets must be taken into account! (<errorID>: 16#xxx2 und 16#xxx5)

The errors “Drive Errors” are a copy of the error bits in the status word of the PSD4xx.

### **Examples**

- Run command (**MC\_Move\_PSD4xx**) with incorrect target value → <errorID> = 16#1008
- Writing a parameter (**MC\_WriteParameter\_PSD4xx**) with invalid parameter number → <errorID> = 16#4020

## 5 Description of the function blocks

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

### 5.1 MC\_Communication\_PSD4xx\_S7-1200\_1500 (FC101), MC\_Communication\_PSD4xx\_S7-300 (FC101)

Use the appropriate function for the following CPU:

CPU	Function
<b>S7-300-CPU</b>	MC_Communication_PSD4xx_S7-300
<b>S7-1200-CPU S7-1500-CPU</b>	MC_Communication_PSD4xx_S7-1200_1500

The functionality is identical in both cases. The **MC\_Communication\_PSD4xx\_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\_PSD4xx
- MC\_ReadParameter\_PSD4xx
- MC\_WriteParameter\_PSD4xx
- MC\_Error\_PSD4xx
- MC\_Parametrization\_PSD4xx
- MC\_PosParametrization\_PSD4xx

The input and output module is stored in the data type <driveData\_PSD4xx>. All other blocks can access it via this interface.

```

19 //function MC_Communication -> communication to PSDs
20 //central communication for all MCs (Motion Controls)
21 "MC_Communication_PSD4xx_S7-1200_1500" (inputAddress := "Data_Store_PSD4xx".PSD_1.pdAddressIn,
22                                     outputAddress := "Data_Store_PSD4xx".PSD_1.pdAddressOut,
23                                     communicationError := "Data_Store_PSD4xx".PSD_1.communicationError,
24                                     drive := "Data_Store_PSD4xx".PSD_1);
--

```

Figure 8: Assignment of variables from DB <Data\_Store> to FC  
**MC\_Communication\_PSD4xx\_1200\_1500**

### 5.2 MC\_Move\_PSD4xx (FB110)

This function block is used to move the drive.

```

26 //"MC_Move_PSD4xx_DB" --> instance of "MC_Move_PSD4xx"
27 "MC_Move_PSD4xx_DB" (execute := "Data_Store_PSD4xx".PSD_Move_1.input.execute,
28                     targetPosition := "Data_Store_PSD4xx".PSD_Move_1.input.targetPosition,
29                     manualRunToLargerValues := "Data_Store_PSD4xx".PSD_Move_1.input.manualRunToLargerValues,
30                     manualRunToSmallerValues := "Data_Store_PSD4xx".PSD_Move_1.input.manualRunToSmallerValues,
31                     timeBusCycle := "Data_Store_PSD4xx".PSD_Move_1.input.timeBusCycle,
32                     active => "Data_Store_PSD4xx".PSD_Move_1.output.active,
33                     done => "Data_Store_PSD4xx".PSD_Move_1.output.done,
34                     actualPosition => "Data_Store_PSD4xx".PSD_Move_1.output.actualPosition,
35                     error => "Data_Store_PSD4xx".PSD_Move_1.output.error,
36                     errorID => "Data_Store_PSD4xx".PSD_Move_1.output.errorID,
37                     drive := "Data_Store_PSD4xx".PSD_1);

```

Figure 9: Assignment of variables from DB <Data\_Store> to FB **MC\_Move\_PSD4xx**

If the drive reports multiple errors, the <errorID> 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#x1xx):

<errorID>	Description
16#x1xx	Error in the state machine (interlock)
16#x2xx	Invalid process data input address
16#x3xx	Invalid process data output address
16#x4xx	Error while reading process data
16#x5xx	Error while writing process data
16#xxx2	Under- or overvoltage motor supply (STO-enabling inactive*)
16#xxx4	Temperature exceeded
16#xxx5	Measuring system error (Measuring system error or STO hardware error*)
16#xxx8	Incorrect target value
16#xxx9	Under- or overvoltage during run / failure voltage control
16#xxx6	Positioning error (block)
16#xxx7	Manual displacement
16#xxxA	Below lower limit
16#xxxB	Upper limit exceeded
16#xxx3	Positioning run aborted
16#xxx1	Drag error

\*If the PSD4xx is a device with STO, then the error description in brackets must be taken into account! (<errorID>: 16#xxx2 und 16#xxx5)

### 5.3 MC\_Error\_PSD4xx (FB111)

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

```

60 // "MC_Error_PSD4xx_DB" --> instance of "MC_Error_PSD4xx"
61 □ "MC_Error_PSD4xx_DB" (execute := "Data_Store_PSD4xx".PSD_Error_1.input.execute,
62 |                               error => "Data_Store_PSD4xx".PSD_Error_1.output.error,
63 |                               errorID => "Data_Store_PSD4xx".PSD_Error_1.output.errorID,
64 |                               errorDescription => "Data_Store_PSD4xx".PSD_Error_1.output.errorDescription,
65 |                               drive := "Data_Store_PSD4xx".PSD_1);

```

Figure 10: Assignment of variables from DB <Data\_Store> to FB **MC\_Error\_PSD4xx**

### 5.4 MC\_Error\_ID\_PSD4xx (FC100)

This FC is used internally as a sub function of the FBs **MC\_Move\_PSD4xx**, **MC\_ReadParameter\_PSD4xx**, **MC\_WriteParameter\_PSD4xx** and **MC\_Error\_PSD4xx**. It just has to be copied from the library into the application program. The setting of the inputs is done completely internally.

## 5.5 MC\_ReadParameter\_PSD4xx (FB112)

This FB can be used to read out parameter values from the drive. All parameters except parameter 23 ("Device type as string") can be read.

```

39 // "MC_ReadParameter_PSD4xx_DB" --> instance of "MC_ReadParameter_PSD4xx"
40 □ "MC_ReadParameter_PSD4xx_DB" (execute := "Data_Store_PSD4xx".PSD_ReadParameter_1.input.execute,
41     parameterNumber := "Data_Store_PSD4xx".PSD_ReadParameter_1.input.parameterNumber,
42     subindex := "Data_Store_PSD4xx".PSD_ReadParameter_1.input.subindex,
43     active => "Data_Store_PSD4xx".PSD_ReadParameter_1.output.active,
44     done => "Data_Store_PSD4xx".PSD_ReadParameter_1.output.done,
45     error => "Data_Store_PSD4xx".PSD_ReadParameter_1.output.error,
46     errorID => "Data_Store_PSD4xx".PSD_ReadParameter_1.output.errorID,
47     value => "Data_Store_PSD4xx".PSD_ReadParameter_1.output.value,
48     drive := "Data_Store_PSD4xx".PSD_1);

```

Figure 11: Assignment of variables from DB <Data\_Store> to FB **MC\_ReadParameter\_PSD4xx**

## 5.6 MC\_WriteParameter\_PSD4xx (FB113)

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

```

50 // "MC_WriteParameter_PSD4xx_DB" --> instance of "MC_WriteParameter_PSD4xx"
51 □ "MC_WriteParameter_PSD4xx_DB" (execute := "Data_Store_PSD4xx".PSD_WriteParameter_1.input.execute,
52     parameterNumber := "Data_Store_PSD4xx".PSD_WriteParameter_1.input.parameterNumber,
53     value := "Data_Store_PSD4xx".PSD_WriteParameter_1.input.value,
54     active => "Data_Store_PSD4xx".PSD_WriteParameter_1.output.active,
55     done => "Data_Store_PSD4xx".PSD_WriteParameter_1.output.done,
56     error => "Data_Store_PSD4xx".PSD_WriteParameter_1.output.error,
57     errorID => "Data_Store_PSD4xx".PSD_WriteParameter_1.output.errorID,
58     drive := "Data_Store_PSD4xx".PSD_1);
59

```

Figure 12: Assignment of variables from DB <Data\_Store> to FB **MC\_WriteParameter\_PSD4xx**

## 5.7 MC\_Parametrization\_PSD4xx (FB114)

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 <parameter\_PSD4xx>.

```

68 // "MC_Parametrization_PSD4xx_DB" --> instance of "MC_Parametrization_PSD4xx"
69 □ "MC_Parametrization_PSD4xx_DB" (execute := "Data_Store_PSD4xx".PSD_Parametrization_1.input.execute,
70     deliveryState := "Data_Store_PSD4xx".PSD_Parametrization_1.input.deliveryState,
71     parameter := "Data_Store_PSD4xx".PSD_Parametrization_1.input.parameter,
72     saveSettings := "Data_Store_PSD4xx".PSD_Parametrization_1.input.saveSettings,
73     active => "Data_Store_PSD4xx".PSD_Parametrization_1.output.active,
74     done => "Data_Store_PSD4xx".PSD_Parametrization_1.output.done,
75     error => "Data_Store_PSD4xx".PSD_Parametrization_1.output.error,
76     errorID => "Data_Store_PSD4xx".PSD_Parametrization_1.output.errorID,
77     errorParameter => "Data_Store_PSD4xx".PSD_Parametrization_1.output.errorParameter,
78     drive := "Data_Store_PSD4xx".PSD_1);
79

```

Figure 13: Assignment of variables from DB <Data\_Store> to FB **MC\_Parametrization\_PSD4xx**

The following must be observed when using the FB:

For each parameter, there is a variable <value> and a variable <enable>. Also see chapter 2.4, parameter\_PSD4xx),

### Example

```
"Data_Store".PSD_Parametrization_1.input.parameter.dirRotation_37.enable = TRUE;
```

```
"Data_Store".PSD_Parametrization_1.input.parameter.dirRotation_37.value = 1;
```

→ Parameter 37 (direction of rotation) is written to counterclockwise.

- By setting <enable> = FALSE, the respective parameter is not written.

- Optionally, a delivery state can be requested before setting individual parameters. To do this, the input <deliveryState> 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 <saveSettings> must be set to TRUE.
- The order of the write accesses is as follows:  
<deliveryState>, parameter 37, 38, 39, 3, 40, 41, 42...and <saveSettings>.
- In the event of a written error, the following parameters are no longer written and the values are not saved if the input <saveSettings> is set.



### INFORMATION!

In case of the S7-300, for the function block **MC\_Parametrization\_PSD4xx** the variable of the type <driveData\_PSD4xx> has to be assigned to the input <driveIn> and to the output <driveOut> (in case of the S7-1200 and S7-1500 in the TIA portal the variable is only connected one time to the Input/Output <Drive>).

## 5.8 MC\_PosParametrization\_PSD4xx (FB115)

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

```

80 // "MC_PosParametrization_PSD4xx_DB" --> instance of "MC_PosParametrization_PSD4xx"
81 □ "MC_PosParametrization_PSD4xx_DB" (execute := "Data_Store_PSD4xx".PSD_PosParametrization_1.input.execute, |
82 | direction := "Data_Store_PSD4xx".PSD_PosParametrization_1.input.direction,
83 | stepsPerTurn := "Data_Store_PSD4xx".PSD_PosParametrization_1.input.stepsPerTurn,
84 | lowerLimit := "Data_Store_PSD4xx".PSD_PosParametrization_1.input.lowerLimit,
85 | upperLimit := "Data_Store_PSD4xx".PSD_PosParametrization_1.input.upperLimit,
86 | setPoint := "Data_Store_PSD4xx".PSD_PosParametrization_1.input.setPoint,
87 | saveSettings := "Data_Store_PSD4xx".PSD_PosParametrization_1.input.saveSettings,
88 | active => "Data_Store_PSD4xx".PSD_PosParametrization_1.output.active,
89 | done => "Data_Store_PSD4xx".PSD_PosParametrization_1.output.done,
90 | error => "Data_Store_PSD4xx".PSD_PosParametrization_1.output.error,
91 | errorID => "Data_Store_PSD4xx".PSD_PosParametrization_1.output.errorID,
92 | errorParameter => "Data_Store_PSD4xx".PSD_PosParametrization_1.output.errorParameter,
93 | drive := "Data_Store_PSD4xx".PSD_1);

```

Figure 14: Assignment of variables from DB <Data\_Store> to FB **MC\_PosParametrization\_PSD4xx**

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) → <direction>
  2. Position scaling numerator (parameter 38) → 400
  3. Position scaling denominator (parameter 39) → <stepsPerTurn>
  4. Actual value (parameter 3) → <setPoint>
  5. If (setPoint > upperLimit):  
Upper mapping end (parameter 41) = <setPoint> + (3 x <stepsPerTurn>)  
Else:  
Upper mapping end (parameter 41) = <upperLimit> + (3 x <stepsPerTurn>)
  6. Upper limit (parameter 42) → <upperLimit>
  7. Lower limit (parameter 43) → <lowerLimit>



### INFORMATION

In case of the S7-300, for the function block **MC\_Parametrization\_PSD4xx** the variable of the type <driveData\_PSD4xx> has to be assigned to the input <driveIn> and to the output <driveOut> (in case of the S7-1200 and S7-1500 in the TIA portal the variable is only connected one time to the Input/Output <Drive>).

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

Condition	<errorID>	<errorParameter>
$\langle \text{stepsPerTurn} \rangle \geq 1$	16#6140	39
$\langle \text{stepsPerTurn} \rangle \leq 10000$	16#6140	39
$\langle \text{lowerLimit} \rangle \leq \langle \text{upperLimit} \rangle$	16#6140	42
$(\langle \text{upperLimit} \rangle - \langle \text{lowerLimit} \rangle) / \langle \text{stepsPerTurn} \rangle \leq 250$	16#6140	43
Falls $\langle \text{setPoint} \rangle < \langle \text{lowerLimit} \rangle$ : $(\langle \text{upperLimit} \rangle - \langle \text{setPoint} \rangle) / \langle \text{stepsPerTurn} \rangle \leq 250$	16#6140	3
Falls $\langle \text{setPoint} \rangle > \langle \text{upperLimit} \rangle$ : $(\langle \text{setPoint} \rangle - \langle \text{lowerLimit} \rangle) / \langle \text{stepsPerTurn} \rangle \leq 250$	16#6140	3

- Optionally at the end, the written values might be saved permanently. To do this, the input <saveSettings> 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 <saveSettings> is set.



## 6 Description of the parameters

### 6.1 <active>

Block	MC_Move_PSD4xx, MC_Parametrization_PSD4xx, MC_PosParametrization_PSD4xx, MC_ReadParameter_PSD4xx, MC_WriteParameter_PSD4xx		
FBs is active or movement order or rather movement is active			
Data type	Bool		
Default	-		
Type	Output		
Description			
<b><u>MC Move PSD4xx</u></b>			
This output is set when			
<ul style="list-style-type: none"><li>the release for moving &lt;execute&gt; is set from FALSE to TRUE</li><li>the release for moving &lt;execute&gt; 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)</li></ul>			
This output is reset when			
<ul style="list-style-type: none"><li>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 6.25 &lt;&lt;<b>timeBusCycle</b>&gt;&gt;) has expired</li><li>a communication error occurs</li></ul>			
<b><u>All other blocks</u></b>			
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 <actualPosition>

Block	MC_Move_PSD4xx	
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 <communicationError>

<b>Block</b>	<b>MC_Communication_PSD4xx...</b>
<b>Communication error to the IO device</b>	
<b>Data type</b>	Bool
<b>Type</b>	Input
<b>Description</b>	
<ul style="list-style-type: none"> <li>The diagnosis of the module states of the IO devices is usually managed centrally in the PLC program, e.g. in OB86 or with the "DeviceStates" instruction.</li> <li>If the corresponding IO device (i.e. the drive in this case) fails, this input must be set to TRUE. As long as communication is not impaired, the input must be assigned FALSE.</li> <li>If there are communication errors, movement order started with <b>MC_Move_PSD4xx</b> are stopped. Likewise, ongoing orders (<b>MC_ReadParameter_PSD4xx</b> and <b>MC_WriteParameter_PSD4xx</b>) are aborted.</li> </ul>	

### 6.4 <deliveryState>

<b>Block</b>	<b>MC_Parametrization_PSD4xx</b>
<b>Loading the factory settings (without saving)</b>	
<b>Data type</b>	Bool
<b>Default</b>	FALSE
<b>Type</b>	Input
<b>Description</b>	
However, the station name and IP address remain unaffected.	

### 6.5 <direction>

<b>Block</b>	<b>MC_PosParametrization_PSD4xx</b>
<b>Direction of rotation of the output shaft</b>	
<b>Data type</b>	Int
<b>Default</b>	0
<b>Type</b>	Input
<b>Description</b>	
<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>	

## 6.6 <done>

Block	MC_Move_PSD4xx, MC_Parametrization_PSD4xx, MC_PosParametrization_PSD4xx, MC_ReadParameter_PSD4xx, MC_WriteParameter_PSD4xx	
Target position reached or FBs is done		
Data type	Bool	
Default	-	
Type	Output	
Description		
<b>MC Move PSD4xx</b> 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 6.25 <timeBusCycle>) after the start by <execute>. If a communication error occurs, it will be reset.		
<b><u>All other blocks</u></b> The bit is set as soon as the process has been successfully completed. It is reset at the start of the respective process.		

## 6.7 <drive>

Block	MC_Communication_PSD4xx..., MC_Error_PSD4xx, MC_Move_PSD4xx, MC_Parametrization_PSD4xx, MC_PosParametrization_PSD4xx, MC_ReadParameter_PSD4xx, MC_WriteParameter_PSD4xx	
Data structure for communication		
Data type	driveData_PSD4xx	
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.		

## 6.8 <error>

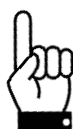
Block	MC_Communication_PSD4xx_..., MC_Error_PSD4xx, MC_Move_PSD4xx, MC_Parametrization_PSD4xx, MC_PosParametrization_PSD4xx, MC_ReadParameter_PSD4xx, MC_WriteParameter_PSD4xx	
FB execution error or drive error		
Data type	Bool	
Default	FALSE	
Type	Output	
Description		
<b><u>MC Move PSD4xx</u></b>		
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).		
<b><u>MC Error PSD4xx</u></b>		
The output <error> is updated every cycle as long as <execute> is set. If <execute> is reset, this output assumes the specified default value.		
<b><u>All other blocks</u></b>		
The error bit is updated every cycle and is set, if an error occurred during an execution of the FB.		

## 6.9 <errorDescription>

Block	MC_Error_PSD4xx	
Error description as text output		
Data type	String	
Default	"	
Type	Output	
Description		
An error description as text output		

## 6.10 <errorID>

Block	MC_Communication_PSD4xx_..., MC_Error_PSD4xx, MC_Move_PSD4xx, MC_Parametrization_PSD4xx, MC_PosParametrization_PSD4xx, MC_ReadParameter_PSD4xx, MC_WriteParameter_PSD4xx	
Error identifier (also see chapter 4 Error description (<errorID>))		
Data type	Word	
Default	0	
Type	Output	
Description		
The error identifier can also be set, while the drive is moving. (e.g. drag error)		
<b><u>MC Move PSD4xx</u></b>		
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).		
<b><u>MC Error PSD4xx</u></b>		
The output <errorID> is updated every cycle as long as <execute> is set. If <execute> is reset, this output assumes the specified default value.		
<b><u>All other blocks</u></b>		
The error identifier bit is updated every cycle and is set, if an error occurred during execution of the FB.		



### ACHTUNG!

The outputs <error> and <errorID> of the function blocks are always updated - even if the <execute> input is not set, except for **MC\_Error\_PSD4xx**.

## 6.11 <errorParameter>

Block	MC_Parametrization_PSD4xx, MC_PosParametrization_PSD4xx	
Parameter number at which the error occurred		
Data type	Int	
Default	0	
Type	Output	
Description		
If there was no error, the value 0 is returned.		

## 6.12 <execute>

Block	MC_Error_PSD4xx, MC_Move_PSD4xx, MC_Parametrization_PSD4xx, MC_PosParametrization_PSD4xx, MC_ReadParameter_PSD4xx, MC_WriteParameter_PSD4xx		
Release of the drive			
Data type	Bool		
Default	FALSE		
Type	Input		
Description			
<b><u>MC Move PSD4xx</u></b>			
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.			
<b><u>All other blocks</u></b>			
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.			

## 6.13 <inputAddress>

Block	MC_Communication_PSD4xx_...	
Address of the input module of the drive		
Data type for S7-1200 S7-1500	Int	
Data type for S7-300	DWord	
Type	Input	
Description		
A detailed description of the address assignment can be found in the chapter 2.1 <driveData_PSD4xx>		

## 6.14 <lowerLimit>

Block	MC_PosParametrization_PSD4xx	
Lower limit		
Data type	DInt	
Default	0	
Type	Input	
Description		
This parameter corresponds to parameter 43 “lower limit”.		

#### 6.15 <manuelRunToLargerValues>

<b>Block</b>	<b>MC_Move_PSD4xx</b>
<b>Manual run to larger values</b>	
<b>Data type</b>	Bool
<b>Default</b>	FALSE
<b>Type</b>	Input
<b>Description</b>	
Manual run is used to increase the values up to the upper limit. The input <execute> must also be set.	

#### 6.16 <manualRunToSmallerValues>

<b>Block</b>	<b>MC_Move_PSD4xx</b>
<b>Manual run to smaller values</b>	
<b>Data type</b>	Bool
<b>Default</b>	FALSE
<b>Type</b>	Input
<b>Description</b>	
Manual run is used to decrease the values down to the lower limit. The input <execute> must also be set.	



#### CAUTION!

When resetting the inputs <manualRunToLargerValues> or <manualRunToSmallerValues>, <execute> must also be reset, otherwise the drive will approach the target position <targetPosition>.

#### 6.17 <outputAddress>

<b>Block</b>	<b>MC_Communication_PSD4xx_...</b>
<b>Address of the output module of the drive</b>	
<b>Data type for S7-1200 S7-1500</b>	Int
<b>Data type for S7-300</b>	DWord
<b>Type</b>	Input
<b>Description</b>	
A detailed description of the address assignment can be found in the chapter 2.1 <driveData_PSD4xx>.	

## 6.18 <parameter>

<b>Block</b>	<b>MC_Parametrization_PSD4xx</b>
<b>Transfer of the parameter set with the data type &lt;parameter_PSD4xx&gt; (also see chapter 2.4 parameter_PSD4xx)</b>	
<b>Data type</b>	parameter_PSD4xx
<b>Default</b>	-
<b>Type</b>	Input
<b>Description</b>	
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 PSD4xx PN.	

## 6.19 <parameterNumber>

<b>Block</b>	<b>MC_WriteParameter_PSD4xx, MC_ReadParameter_PSD4xx</b>
<b>Parameter number of the parameter to be read or written</b>	
<b>Data type</b>	Int
<b>Default</b>	0
<b>Type</b>	Input
<b>Description</b>	
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.	

## 6.20 <saveSettings>

<b>Block</b>	<b>MC_Parametrization_PSD4xx, MC_PosParametrization_PSD4xx</b>
<b>Saving the parameter settings</b>	
<b>Data type</b>	Bool
<b>Default</b>	FALSE
<b>Type</b>	Input
<b>Description</b>	
This parameter corresponds to parameter 96 „delivery state. (function <Writing "1">)	



### 6.21 <setPoint>

<b>Block</b>	<b>MC_PosParametrization_PSD4xx</b>
<b>Setpoint of the measuring system</b>	
<b>Data type</b>	DInt
<b>Default</b>	0
<b>Type</b>	Input
<b>Description</b>	
This parameter corresponds to parameter 3 "actual value".	

### 6.22 <stepsPerTurn>

<b>Block</b>	<b>MC_PosParametrization_PSD4xx</b>
<b>Steps per rotation on the output shaft (resolution)</b>	
<b>Data type</b>	Int
<b>Default</b>	0
<b>Type</b>	Input
<b>Description</b>	
The number of steps per rotation <stepsPerTurn> 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).	

### 6.23 <subIndex>

<b>Block</b>	<b>MC_ReadParameter_PSD4xx</b>
<b>Subindex of the parameters</b>	
<b>Data type</b>	Int
<b>Default</b>	0
<b>Type</b>	Input
<b>Description</b>	
The value of the parameter <subIndex> can be left at the default value. Other values are currently not planned.	

## 6.24 <targetPosition>

<b>Block</b>	<b>MC_Move_PSD4xx</b>
<b>Target position to be approached</b>	
<b>Data type</b>	DInt
<b>Default</b>	0
<b>Type</b>	Input
<b>Description</b>	
If a new target position is set during a run, it is approached immediately. If <execute> 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 <execute> must be reset and set again.

## 6.25 <timeBusCycle>

<b>Block</b>	<b>MC_Move_PSD4xx</b>
<b>Cycle time of the Profinet bus cycle</b>	
<b>Data type</b>	Time
<b>Default</b>	40ms
<b>Type</b>	Input
<b>Description</b>	
<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 &lt;timeBusCycle&gt; has been implemented. 40 ms are set as the initial value for a bus cycle. For this duration, the output &lt;active&gt; is always set and the output &lt;done&gt; 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 &lt;timeBusCycle&gt; can also be reduced.</p>	

## 6.26 <upperLimit>

Block	MC_PosParametrization_PSD4xx	
Upper limit		
Data type	DInt	
Default	0	
Type	Input	
Description		
This parameter corresponds to parameter 42 “upper limit”.		

## 6.27 <value>

Block	MC_ReadParameter_PSD4xx, MC_WriteParameter_PSD4xx	
value to be written or read value of a parameter		
Data type	DInt	
Default	0	
Type	Input for <b>MC_WriteParameter_PSD4xx</b> Output for <b>MC_ReadParameter_PSD4xx</b>	
Description		
A read or write 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 TIA versions can be done without any problems.

There are two ways of using the function blocks:

- Project
- Library

### 7.1 Project

Opening and upgrading the project depending on the CPU:

- example\_hw\_PSD4xx\_1200\_1500\_xxxxxxx.zap13  
→ for SIMATIC S7 - 1200 and SIMATIC S7 – 1500
- example\_hw\_PSD4xx\_300\_xxxxxxx.zap13  
→ for SIMATIC S7 - 300

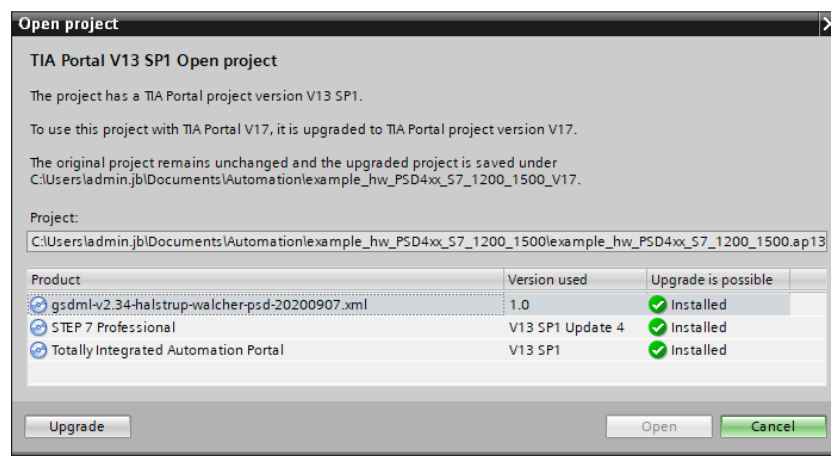


Figure 15: Upgrade of a project in TIA V13 to TIA V17

### 7.2 Library

Opening and upgrading the library depending on the CPU:

- library\_PSD4xx\_1200\_1500\_xxxxxxx.zal13  
→ for SIMATIC S7 - 1200 and SIMATIC S7 – 1500
- library\_PSD4xx\_300\_xxxxxxx.zal13  
→ for SIMATIC S7 – 300

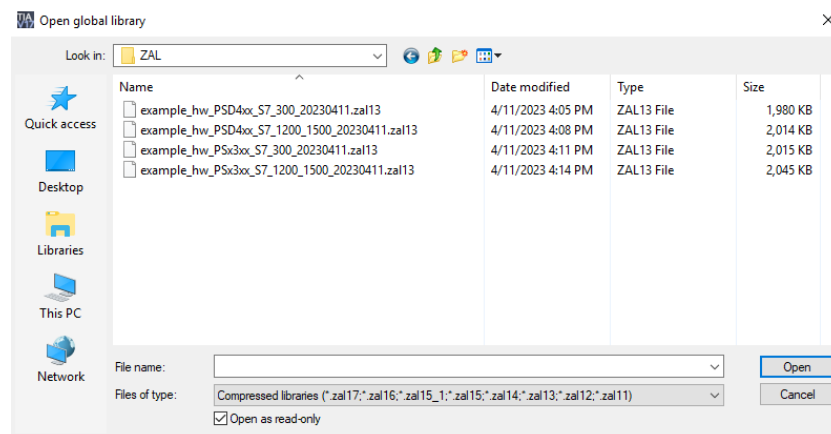


Figure 16: Opening a compressed library

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

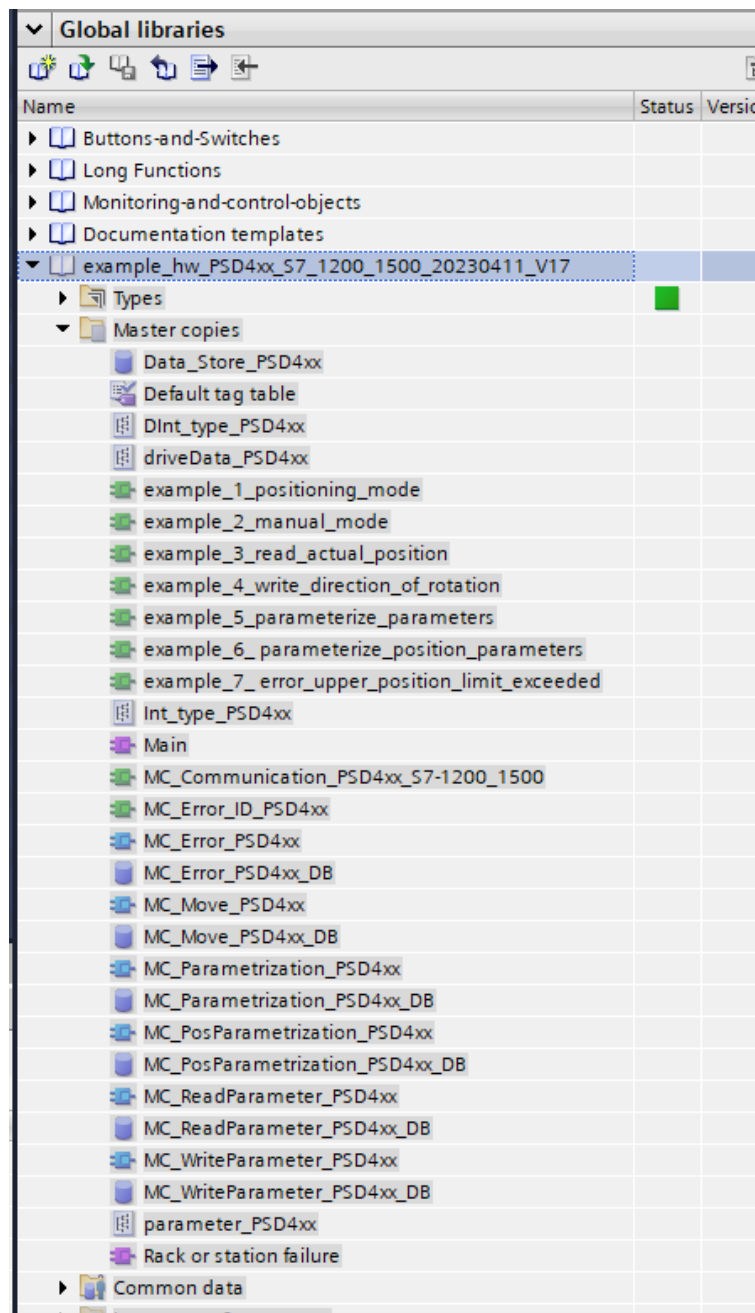


Figure 17: Global library in TIA V17

The following functions from the library must always be copied into the user's project (regardless of the function blocks **MC\_...\_PSD4xx** actually used and the number of drives):

- for SIMATIC S7 - 1200 and S7 - 1500: **MC\_Communication\_PSD4xx\_S7-1200\_1500**  
for SIMATIC S7 - 300: **MC\_Communication\_PSD4xx\_S7-300**
- **MC\_Error\_ID\_PSD4xx**

These functions are used for communication and error detection with the drive.

In addition, the required function blocks **MC\_...\_PSD4xx** must be copied into the user project. All or a selection of the following blocks can be used:

- **MC\_Move\_PSD4xx**
- **MC\_Error\_PSD4xx**
- **MC\_ReadParameter\_PSD4xx**
- **MC\_WriteParameter\_PSD4xx**
- **MC\_Parametrization\_PSD4xx**
- **MC\_PosParametrization\_PSD4xx**
- Data type <Int\_type\_PSD4xx> (for **MC\_Parametrization\_PSD4xx**)
- Data type <DInt\_type\_PSD4xx> (for **MC\_Parametrization\_PSD4xx**)
- Data type <parameter\_PSD4xx> (for **MC\_Parametrization\_PSD4xx**)

In addition, the data block <Data\_Store> can be used as a template for the assignment to the function blocks. In its standard version, the data types <Int\_type\_PSD4xx>, <DInt\_type\_PSD4xx> and <parameter\_PSD4xx> (for **MC\_Parametrization\_PSD4xx**) must then also be copied.

### 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.

There are following rules:

- If the input <execute> of **MC\_Move\_PSD4xx** is set, the function blocks **MC\_Parametrization\_PSD4xx** und **MC\_PosParametrization\_PSD4xx** cannot be activated (<errorID>: 16#x100).
- On the other hand, it is possible to activate **MC\_ReadParameter\_PSD4xx** or **MC\_WriteParameter\_PSD4xx** during a movement (e.g. to read out the current torque or to change the target speed during the actual movement).
- If the input <execute> of **MC\_Parametrization\_PSD4xx** or **MC\_PosParametrization\_PSD4xx** is set, the function block **MC\_Move\_PSD4xx** cannot be activated (<errorID>: 16#0100).
- Only one of the function blocks **MC\_ReadParameter\_PSD4xx**, **MC\_WriteParameter\_PSD4xx**, **MC\_Parametrization\_PSD4xx** or **MC\_PosParametrization\_PSD4xx** can be active at the same time. If the input <execute> of a function block is set and the input <execute> of another function block is set, then there is the <errorID> 16#x100.
- The function block **MC\_Error\_PSD4xx** can always be activated.

### 7.4 Several PSD4xx in a project

There is only one PSD4xx in the example projects. If several drives are used, the project must be adjusted accordingly.

For example for three PSD4xx:

1. Initialize instance data blocks
  - three instance data blocks of type FB110 (**MC\_Move\_PSD4xx**), e.g.
    - DB110<sup>1</sup>, symbolic name = MC\_Move\_PSD4xx\_DB
    - DB110<sup>2</sup>, symbolic name = MC\_Move\_PSD4xx\_DB
    - DB110<sup>3</sup>, symbolic name = MC\_Move\_PSD4xx\_DB
  - three instance data blocks of type FB111 (**MC\_Error\_PSD4xx**), e.g.
    - DB111<sup>1</sup>, symbolic name = MC\_Error\_PSD4xx\_DB
    - DB111<sup>2</sup>, symbolic name = MC\_Error\_PSD4xx\_DB
    - DB111<sup>3</sup>, symbolic name = MC\_Error\_PSD4xx\_DB
  - further required instance data blocks

2. Afterward, the data block of the type "global" with the name DB100 (<Data\_Store\_PSD4xx>) is then adapted. Three variables of the <driveData\_PSD4xx> type are created there, e.g. with the following designations:
  - "PSE\_1"
  - "PSE\_2"
  - "PSE\_3"
3. In addition, the variables must be created in <Data\_Store\_PSD4xx> to control the individual blocks:
  - Three variables for FB **MC\_Move\_PSD4xx**, e.g.
    - "PSE\_Move\_1"
    - "PSE\_Move\_2"
    - "PSE\_Move\_3"
  - Three variables for FB **MC\_Error\_PSD4xx**, e.g.
    - "PSE\_Error\_1"
    - "PSE\_Error\_2"
    - "PSE\_Error\_3"
  - Additional variables
4. Finally, the variables from DB <Data\_Store\_PSD4xx> must be assigned to the function blocks (also see chapter 0)

## 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.

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
(* MAIN *)

//call of function
"example_1_positioning_mode"();

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

```

Figure 18: Calling function in the main program

### Similarity of all example programs

- In each example program, the function **MC\_Communication\_PSD4xx\_S7-1200\_1500** called cyclically at the beginning.
- For all functions, the required instances are defined and called at the beginning.
- For all functions, the variable <startProgram> must be set to TRUE to start the program.
- 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 <stopProgram>.

### 8.1 Example 1: positioning mode

The positioning mode of the **FB MC\_Move\_PSD4xx** is presented in this example program. In this example, the drive moves an endless loop between the values 60000 and 50000.

#### Sequence of the program

Case	Description
0	All relevant variables are set to FALSE for initialization and the index is incremented by one. (It always makes sense to set all relevant variables to the default value at the beginning of a program.)
1	The target position of 60000 and the move command are set. If the current position is at 60000±2 (also see parameter 44 "positioning window") and the movement was successfully completed, then the movement command is reset and the index is increased by one.
2	The target position of 50000 and the move command are set. If the current position is at 50000±2 (also see parameter 44 "positioning window") and the movement was successfully completed, then the movement command is reset and the index is set to 1.



## 8.2 Example 2: manual mode

The manual mode of the FB **MC\_Move\_PSD4xx** is presented in this example program. 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 index is incremented by one. (It always makes sense to set all relevant variables to the default value at the beginning of a program.)
1	The target position of 55000 and the move command are set. If the actual position is at $55000 \pm 2$ (also see parameter 44 "positioning window") and the movement was successfully completed, then the movement command is reset and the index is increased by one.
2	With manual mode, the target position of 60000 is approached. If the actual value corresponds to 60000 then the movement command for manual movement is reset and the index is set to 100.

## 8.3 Example 3: read actual value

The FB **MC\_ReadParameter\_PSD4xx** is presented in this example program. 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 index is incremented by one. (It always makes sense to set all relevant variables to the default value at the beginning of a program.)
1	The target position of 55000 and the move command are set. If the actual position is at $55000 \pm 2$ (also see parameter 44 "positioning window") and the movement was successfully completed, then the movement command is reset and the index is increased by one.
2	The parameter "actual value" is read. If the value is $55000 \pm 2$ (also see parameter 44 "positioning window"), then the read command is reset and the index is set to 100.

#### 8.4 Example 4: write direction of rotation

The FB **MC\_WriteParameter\_PSD4xx** is presented in this example program. In this example, the parameter "direction of rotation" is written and read and positioning moves are made.

##### Sequence of the program

Case	Description
0	All relevant variables are set to FALSE for initialization and the index is incremented by one. (It always makes sense to set all relevant variables to the default value at the beginning of a program.)
1	The target position of 60000 and the move command are set. If the actual position is at $60000 \pm 2$ (also see parameter 44 "positioning window") and the movement was successfully completed, then the movement command is reset and the index is increased by one.
2	The value of the parameter "direction of rotation" is written to 1. If the write command was successfully completed, then the write command is reset and the index is increased by one.
3	The value of the parameter "direction of rotation" is read. It is checked, if the value of the parameter "direction of rotation" is 1. If that is the case and the read command was successfully completed, then the read command is reset and the index is increased by one.
4	The change in direction of rotation causes, that the actual position is at 42400 ( $102400 - 60000 = 42400$ ). The target position of 60000 and the move command are set. If the actual value is $60000 \pm 2$ (see parameter 44 "positioning window") and the movement were successfully completed, then the movement command is reset and the index is increased by one.
5	The value of the parameter "direction of rotation" is written to 0. If the write command was successfully completed, then the write command is reset and the index is increased by one.
6	The value of the parameter "direction of rotation" is read. It is checked, if the value of the parameter "direction of rotation" is 0. If that is the case and the read command was successfully completed, then the read command is reset and the index is increased by one.
7	The change in direction of rotation causes, that the actual position is at 42400 ( $102400 - 60000 = 42400$ ). The target position of 60000 and the move command are set. If the actual value is $60000 \pm 2$ (see parameter 44 "positioning window") and the movement were successfully completed, then the movement command is reset and the index is set to 100.

## 8.5 Example 5: parameterize parameters

The FB **MC\_Parametrization\_PSD4xx** is presented in this example program. In this example, the parameters "loop length" and "target speed" are parameterize and then a positioning move are made.

### Sequence of the program

Case	Description
0	All relevant variables are set to FALSE for initialization and the index is incremented by one. (It always makes sense to set all relevant variables to the default value at the beginning of a program.)
1	The target position of 60000 and the move command are set. If the actual position is at $60000 \pm 2$ (also see parameter 44 "positioning window") and the movement was successfully completed, then the movement command is reset and the index is increased by one.
2	The parameters "loop length" and "target speed" are parameterized. The <enable> of the parameters must be set. If the parameterization command was successfully completed, then the parameterization command is reset and the index is increased by one.
3	The target position of 50000 and the move command are set. Here there is a positioning move without loop and it is a move with 20 rpm. If the actual position is at $55000 \pm 2$ (also see parameter 44 "positioning window") and the movement was successfully completed, then the movement command is reset and the index is set to 100.

## 8.6 Example 6: parameterize position parameters

The FB **MC\_PosParametrization\_PSD4xx** is presented in this example program. Parameters are set 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 index is incremented by one. (It always makes sense to set all relevant variables to the default value at the beginning of a program.)
1	The target position of 50000 and the move command are set. If the actual position is at $50000 \pm 2$ (also see parameter 44 "positioning window") and the movement was successfully completed, then the movement command is reset and the index is increased by one.
2	All parameters affecting the position are parameterized. The parameter "actual position" is set to 0, the parameter "upper limit" to 10000 and the parameter "lower limit" to -10000. The rest of the parameters are not changed and the parameter values should not be saved. If the parameterization command for the position parameters was successfully completed, then the parameterization command for the position parameters is reset and the index is increased by one.
3	The parameter "actual value" is read. If the value is 0 (in this case the actual value must be exactly 0), then the read command is reset and the index is set to 100.

## 8.7 Example 7: error upper limit exceeded

The FB **MC\_Error\_PSD4xx** is presented in this example program. It is moved to the upper limit and the error description is read out.

### Sequence of the program

Case	Description
0	All relevant variables are set to FALSE for initialization and the index is incremented by one. (It always makes sense to set all relevant variables to the default value at the beginning of a program.)
1	The target position of 65000 and the move command are set. If the actual position is at $65000 \pm 2$ (also see parameter 44 "positioning window") and the movement was successfully completed, then the movement command is reset and the index is increased by one.
2	The value of the parameter "upper limit" is written to 70000. If the write command was successfully completed, then the write command is reset and the index is increased by one.
3	The FB <b>MC_Error_PSD4xx</b> is activated to detect errors.
4	The upper limit is approached in manual mode. Only when the actual position is 70000, the error description is displayed as a string. Then the drive command for manual run is reset and the index is set to 100. (It would also be possible to get this error from the <errorID> output of the FB <b>MC_Move_PSD4xx</b> . The <errorID> would be 0x100B.)

## 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: Data type <Int_type_PSD4xx>.....	7
Figure 4: Data type <DInt_type_PSD4xx> .....	7
Figure 5: Extract of data type <parameter_PSD4xx>.....	8
Figure 6: Assignment of variables from DB <Data_Store> to FB <b>MC_Move_PSD4xx</b> .....	9
Figure 7: DB <Data_Store> with the variables for one device.....	9
Figure 8: Assignment of variables from DB <Data_Store> to FC	
<b>MC_Communication_PSD4xx_1200_1500</b> .....	12
Figure 9: Assignment of variables from DB <Data_Store> to FB <b>MC_Move_PSD4xx</b> .....	12
Figure 10: Assignment of variables from DB <Data_Store> to FB <b>MC_Error_PSD4xx</b> .....	13
Figure 11: Assignment of variables from DB <Data_Store> to FB <b>MC_ReadParameter_PSD4xx</b> .....	14
Figure 12: Assignment of variables from DB <Data_Store> to FB <b>MC_WriteParameter_PSD4xx</b> .....	14
Figure 13: Assignment of variables from DB <Data_Store> to FB <b>MC_Parametrization_PSD4xx</b> .....	14
Figure 14: Assignment of variables from DB <Data_Store> to FB <b>MC_PosParametrization_PSD4xx</b>	
.....	15
Figure 15: Upgrade of a project in TIA V13 to TIA V17 .....	28
Figure 16: Opening a compressed library.....	28
Figure 17: Global library in TIA V17 .....	29
Figure 18: Calling function in the main program.....	32