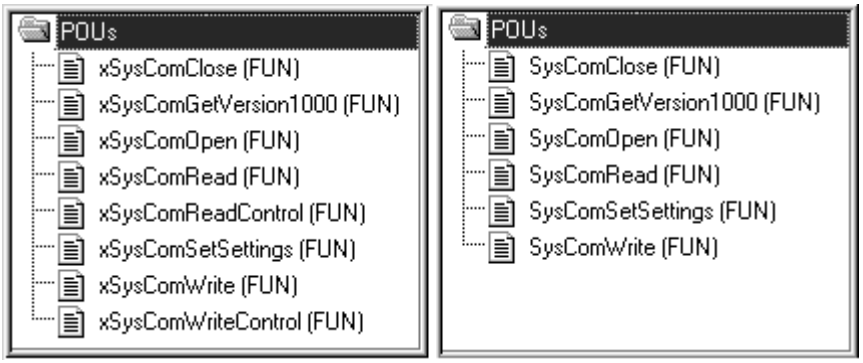
10/10 MN05010002Z-EN
replaces 02/05 AWB2786-1456GB

Function Blocks for CoDeSys



Powering Business Worldwide

Emergency On Call Service
Please call your local representative:
http://www.eaton.com/moeller/aftersales
or
Hotline After Sales Service:
+49 (0) 180 5 223822 (de, en)
AfterSalesEGBonn@eaton.com

Original Operating Instructions
The German-language edition of this document is the original operating manual.

Translation of the original operating manual
All editions of this document other than those in German language are translations of the original German manual.

1st published 2002, edition date 08/02
2nd edition 11/2002
3rd edition 04/2003
4th edition 04/2004
5th edition 07/2004
6th edition 09/2004
7th edition 02/2005
8th edition 10/2010
See revision protocol in the "About this manual" chapter

Authors: Norbert Mausolf, Peter Roersch, Oswald Weiß
Editor: Thomas Kracht
Translator: Patrick Chadwick

Contents

About this manual

List of revisions

Edition date	Chapter	Subject	New
04/03	chapter8	„Data access function blocks for files: XC100_File.lib“	×
04/04	chapter6	„Counter elements: Counter_Analog.lib (for XIOC-2CNT-2AO-NC)“	×
07/04	chapter9	„Acyclic file access blocks for PROFIBUS-DP: xSysNetDPMV1.lib“	×
09/04	chapter10	„Communication block for Suconet K slaves: SuconetK.lib“	×
02/05	chapter2	„Date and Time function blocks of the S40: XS40_MoellerFB_RTC.lib“	×
	page 12	„CounterFunctionBlocks“	×
	page 18	„RegisterFunctionBlocks“	×
	chapter11	„Communication block for Suconet K slaves: SuconetK.lib“	×
	chapter12	„Diagnostics function blocks: XSysDiagLib.lib“	×
	chapter13	„Transparent mode functions: xSysCom200/ SysLibCom/ XC100_SysLibCom/XN-PLC-SysLibCom.lib“	
10/10	All	Change to Eaton terminolgy	×

Library summary

The function blocks described here are compiled in the programming software in libraries in accordance with their functions. The following libraries are available:

Table 1: Library summary

Library name	Type of function block
XS40_MoellerFB.lib	Standard function blocks for the S40 software
XS40_MoellerFB_RTC.lib	Clock function blocks of the S40
RTCLib.lib	Clock function blocks
Visu.lib	Visualization function blocks
Counter.lib	Counter function blocks for counter modules
Counter_Analog.lib	Counter function blocks for XIOC-2CNT-2AO-NC
CANopen_Utilities.lib	Transfer function block
Standard.lib	IEC function blocks and functions[1)]
XC100_File.lib	Data access function blocks
xSysNetDPVM1.lib	Acyclic data access function blocks for PROFIBUS-DP
SuconetK.lib	Communication function block for Suconet-K slaves

1) They are described in the manual for programming software (MN05010003Z-EN; previously AWB2700-1437GB).

The description of the interfaces for parameter transfer is made on the basis of prototypes for the individual function blocks.
A prototype provides an overview of the inputs and outputs of a function block. In the IL representation, the prototype is the declaration portion of the function block, which includes the declaration blocks. A graphic prototype presentation is used in the description of the function blocks, to enhance clarity. The function block is depicted as a rectangular circuit symbol, with the inputs on the left side and the outputs on the right side.

The name of the function block is placed in the centre of the circuit symbol. The data types for the inputs and outputs are shown outside the symbol.

The symbol “>” in front of an input operand indicates that the input requires a rising edge (positive-going transition) to initiate the function.

Additional manuals

The user interface of the programming software is to be found in the manual MN05010003Z-EN (previously AWB2700-1437GB). There are separate tool boxes for the "Control Engineering" and "Motion Control" fields of application. The function blocks which are included are described in the manuals MN05010004Z-EN (previously AWB2700-1451GB) (Closed-Loop Control Toolbox) and MN05010005Z-EN (previously AWB2700-1454GB) (Motion Control Toolbox).

These manuals are available online, as a PDF file, at:
http://www.eaton.com/moeller → **Support**
Enter the manual number here as the search text.

Symbols

The symbols used in this manual have the following meanings:

▶ Indicates instructions for user actions.

→ Indicates useful advice and additional information.

Important
Indicates a risk of material damage.

Caution!
Indicates the risk of major damage to property, or slight injury.

Warning!
Indicates a risk of serious material damage or serious or fatal injury.

For greater clarity, the name of the current chapter is shown in the header of the left-hand page and the name of the current section in the header of the right-hand page. Exceptions are the first page of each chapter, and empty pages at the end.

1 Standard S40 function blocks: XS40_MoellerFB.lib

The library contains the following function blocks:

- Communication
  - MI4netDP16
  - MI4netDP32
  - MV4netDP38
  - MV4netDP70

- Convert
  - DataScale
  - IEEE_To_Real
  - Real_To_IEEE

- CounterFunctionBlocks
  - S40_16BitCounter
    S40_32BitCounter
    Up/down counter

- DateAndTime
  - DATConcatX
  - DateConcat
  - DateSplit
  - DATSplitX
  - TimeConcatX
  - TimeSplitX
  - TODconcat
  - TODsplit

- RegisterFunctionBlocks
  - FifoBx
  - FifoWx
  - LifoBx
  - LifoWx
  - SR_x
  - SRB_x
  - SRW_x

- Timer
  - MS_TimeFalling
  - MS_TimeRising
  - S_TimeFalling
  - S_TimeRising
  - TimeGenerator
  - TimePulse

Sucosoft S40 compatibility

The function blocks described here differ from the corresponding function blocks of Sucosoft S40 as described below:

Function blocks from the "Communication" section
Function blocks from the "Communication" section (MI4/MV4 drivers for linking to PROFIBUS-DP) have identical functions to the S40 OEM-function block "MI4netK".

The alterations are:

- the name of the parameters
  - b_ar_IB instead of RDB_Start
  - b_ar_QB instead of SDB_Start and
  - b_Status instead of Status

- arrays of type VAR_IN_OUT have fixed lengths. The length that is to be set is already given in the name of the function block (e.g. MI4netDP32 -> ARRAY[0..31]) and is independent of the input/output module that was selected in the controller configuration.

Function blocks from the "Timer" section
The function blocks from the "Timer" section are identical to the corresponding S40 function blocks.

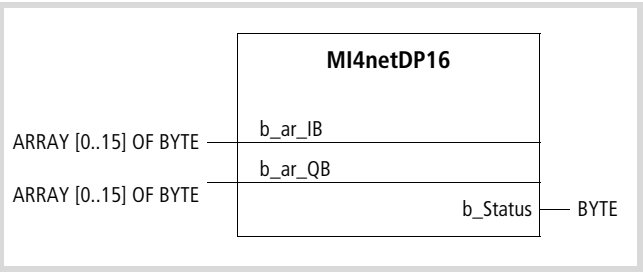
Function blocks from the "DateAndTime" section
Function blocks from the "DateAndTime" section that have an "X" at the end of the name differ from the corresponding S40 functions as follows:

DATconcatX	No "MilliSecond" input
DATsplitX	No "MilliSecond" output
TimeConcatX	No "Sign" input
	Maximum timing range is T#49d17h2m47s295ms
	Additional output "OverflowCounter" -> indicates the number of overflows during the maximum timing range.
TimeSplitX	No "Sign" output
	Maximum timing range is T#49d17h2m47s295ms; if an overrange occurs, the data type TIME is changed to the remainder value, without any error message.

The other function blocks from the "Date and Time" section are identical to the corresponding S40 function blocks.

Communication

MI4netDP16
communication block for MI4
(16 byte PROFIBUS-DP data channel)



Function block prototype

Meanings of the operands

b_ar_IB	PROFIBUS-DP data transfer interface for the input data (Input), e.g. with start address 0: ARRAY [0..15] OF BYTE AT %IB0
b_ar_QB	PROFIBUS-DP data transfer interface for the output data (Output), e.g. with start address 0: ARRAY [0..15] OF BYTE AT %QB0
b_Status	Communication status MI4

→ In the controller configuration, select the 16-byte input/output module of the MI4-PROFIBUS DP interface module ZB4-504-IF1 or ZB4-504-IF2!

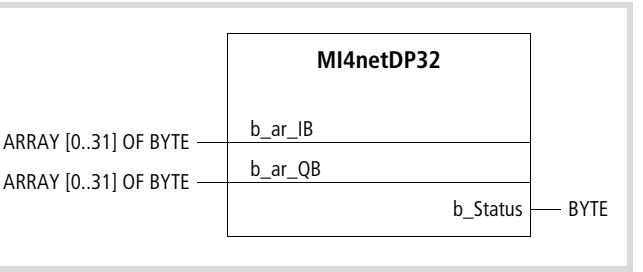
Description
"MI4netDP16" is a communication block for linking the M14 visualisation devices to an XC controller via PROFIBUS-DP.

The function block copies the data transported through the 16-byte PROFIBUS-DP data channel to the physical marker addresses %M.. , which you have already configured in MI4.

"b_Status" provides information and error messages concerning communication:

01h:	XC sends data packet to MI4
02h:	XC receives data packet from MI4
04h:	no data request
80h:	MI4 sends invalid operation code/communication error
81h:	Marker range overflow/telegram length error on reading markers from the XC
82h:	Marker range overflow/telegram length error on writing the markers in the XC

MI4netDP32
communication block for MI4
(32 byte PROFIBUS-DP data channel)



Function block prototype

Meanings of the operands

b_ar_IB	PROFIBUS-DP data transfer interface for the input data (Input), e.g. with start address 0: ARRAY [0..31] OF BYTE AT %IB0
b_ar_QB	PROFIBUS-DP data transfer interface for the output data (Output), e.g. with start address 0: ARRAY [0..31] OF BYTE AT %QB0
b_Status	Communication status MI4

→ In the controller configuration, select the 32-byte input/output module of the MI4-PROFIBUS DP interface module ZB4-504-IF1 or ZB4-504-IF2!

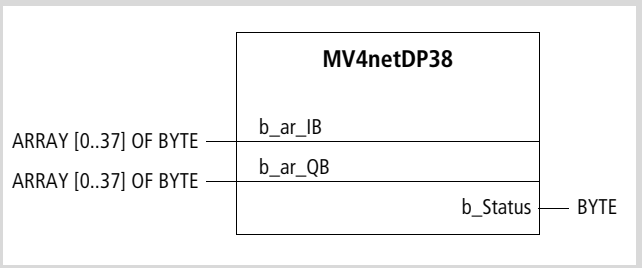
Description
"MI4netDP32" is a communication block for linking the M14 visualisation devices to an XC controller via PROFIBUS-DP.

The function block copies the data transported through the 32-byte PROFIBUS-DP data channel to the physical marker addresses %M.. , which you have already configured in MI4.

"b_Status" provides information and error messages concerning communication:

01h:	XC sends data packet to MI4
02h:	XC receives data packet from MI4
04h:	no data request
80h:	MI4 sends invalid operation code/communication error
81h:	Marker range overflow/telegram length error on reading markers from the XC
82h:	Marker range overflow/telegram length error on writing the markers in the XC

MV4netDP38
communication block for MV4
(38 byte PROFIBUS-DP data channel)



Function block prototype

Meanings of the operands

b_ar_IB	PROFIBUS-DP data transfer interface for the input data (Input), e.g. with start address 0: ARRAY [0..37] OF BYTE AT %IB0
b_ar_QB	PROFIBUS-DP data transfer interface for the output data (Output), e.g. with start address 0: ARRAY [0..37] OF BYTE AT %QB0
b_Status	Communication status

→ In the controller configuration, select the 38-byte input/output module of the MV4-PROFIBUS DP interface module ZB4-604-IF1 or ZB4-IF2!

Description

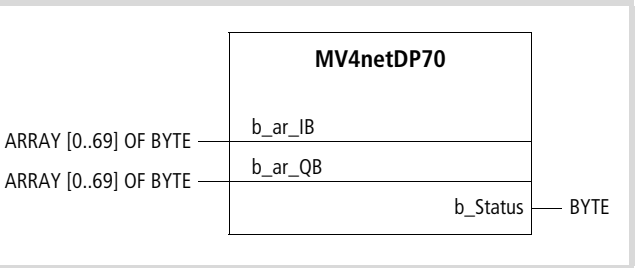
"MV4netDP38" is a communication block for linking the MV4 visualisation devices to an XC controller via PROFIBUS-DP.

The function block copies the data transported through the 38-byte PROFIBUS-DP data channel to the physical marker addresses %M.. , which you have already configured in MV4.

"b_Status" provides information and error messages concerning communication:

01h:	XC sends data packet to MV4
02h:	XC receives data packet from MV4
04h:	no data request
80h:	MV4 sends invalid operation code/communication error
81h:	Marker range overflow/telegram length error on reading markers from the XC
82h:	Marker range overflow/telegram length error on writing the markers in the XC

MV4netDP70
communication block for MV4
(70 byte PROFIBUS-DP data channel)



Function block prototype

Meanings of the operands

b_ar_IB	PROFIBUS-DP data transfer interface for the input data (Input), e.g. with start address 0: ARRAY [0..69] OF BYTE AT %IB0
b_ar_QB	PROFIBUS-DP data transfer interface for the output data (Output), e.g. with start address 0: ARRAY [0..69] OF BYTE AT %QB0
b_Status	Communication status

→ In the controller configuration, select the 70-byte input/output module of the MV4-PROFIBUS DP interface module ZB4-604-IF1 or ZB4-IF2!

Description

"MV4netDP70" is a communication block for linking the MV4 visualisation devices to an XC controller via PROFIBUS-DP.

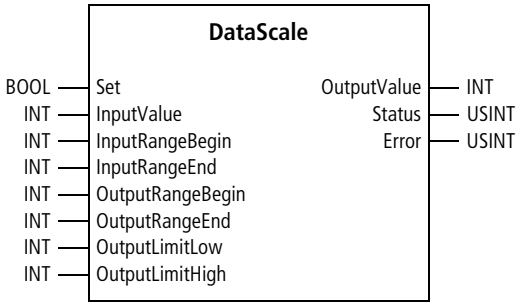
The function block copies the data transported through the 70-byte PROFIBUS-DP data channel to the physical marker addresses %M.. , which you have already configured in MV4.

"b_Status" provides information and error messages concerning communication:

01h:	XC sends data packet to MV4
02h:	XC receives data packet from MV4
04h:	no data request
80h:	MV4 sends invalid operation code/communication error
81h:	Marker range overflow/telegram length error on reading markers from the XC
82h:	Marker range overflow/telegram length error on writing the markers in the XC

Convert

DataScale
Data scaling



Function block prototype

Meanings of the operands

Set	Activate the function block
InputValue	Input value (IV), which can vary within the input range; IRB ≤ IV ≤ IRE
InputRangeBegin	Input range begin (IRB) IRB < IRE
InputRangeEnd	Input range end (IRE)
OutputRangeBegin	Output range begin (ORB) ORB < ORE
OutputRangeEnd	Output range end (ORE)
OutputLimitLow	Output low limit (OLL) OLL < OLH ≤ ORE and OLL ≥ ORB
OutputLimitHigh	Output high limit (OLH) OLH > OLL ≥ ORB and OLH ≤ ORE
OutputValue	Output value
Status	Status messages (range infringement)
Error	Error messages (parameters)

Description

The values IRE (largest input value) and IRB (smallest input value) define an input range within which the value to be processed (IV) can vary. This input range can be assigned to a "free" output range (ORE and ORB). The input value is then converted according to the ratio of the input and output ranges. Two limit values (OLL and OLH) must be defined for this output range. A message will be generated if the converted value goes above or below these limits.

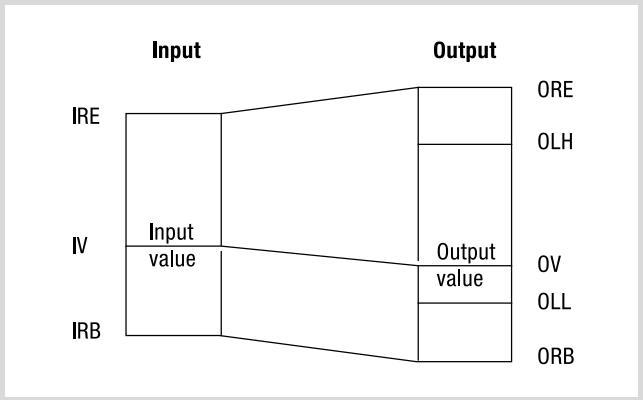


Figure 1: Definition of the input/output range

A logic "1" signal at the "Set" input tests the range parameter values that have been set by the user. Wrong values are indicated by the output byte "Error". The value appearing at the input ("InputValue") is converted and presented at the output ("OutputValue").

If the "IRB" or "IRE" limits are infringed, then the output "OutputValue" is set for the affected limit, and the error bits are set for the "Status" output.

If there is no error present, the conversion continues as long as the "1" signal is present at the "Set" input.

Alterations to the range values only become effective after the "Set" input has been set to "0" and then to "1" again.

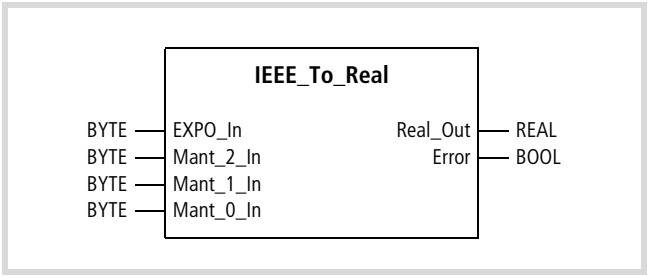
If a "0" signal is present at the "Set" input, then the outputs are: "OutputValue" = ORB, "Error" = 0 and "Status" = 0.

Error (value)	Error messages (parameters)
0	No error
1	IRB ≧ IRE
2	ORB ≧ ORE
3	OLL > OLH, OLL < ORB, OLL > ORE, OLH < ORB, OLH > ORE

Status (value)	Messages (range infringements)
0	No error
1	OV < OLL
2	OV > OLH
3	IV < IRB
4	IV > IRE

When the error is cleared, the byte is reset to 0.

IEEE_To_Real
number conversion: IEEE-754 standard format to data type REAL



Function block prototype

Meanings of the operands

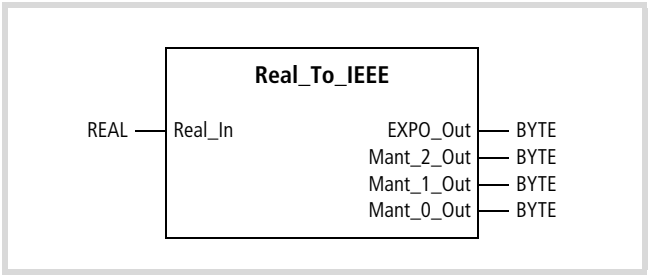
EXPO_In	IEEE float number for conversion: exponent (incl. sign)
Mant_2_In	IEEE float number for conversion: mantissa byte 2 (incl. 1 bit exponent)
Mant_1_In	IEEE float number for conversion: mantissa byte 1
Mant_0_In	IEEE float number for conversion: mantissa byte 0
Real_Out	Converted real number
Error	Conversion error: IEEE number cannot be displayed

Description
The function block "IEEE_To_Real" converts a floating point number in the IEEE-754 single-precision format into a number of data type REAL. The representation of the float number requires four bytes (32 bits). This conversion changes the format only, not the value:

EXPO_In	Mant_2_In	Mant_1_In	Mant_0_In

The IEEE format is mainly used for transferring floating point numbers to other systems (e.g. via PROFIBUS).

Real_To_IEEE
number conversion: data type REAL to IEEE-754 standard format



Function block prototype

Meanings of the operands

Real_In	Real number for conversion
EXPO_Out	IEEE float number: exponent (incl. sign)
Mant_2_Out	IEEE float number: mantissa byte 2 (incl. 1 bit exponent)
Mant_1_Out	IEEE float number: mantissa byte 1
Mant_0_Out	IEEE float number: mantissa byte 0

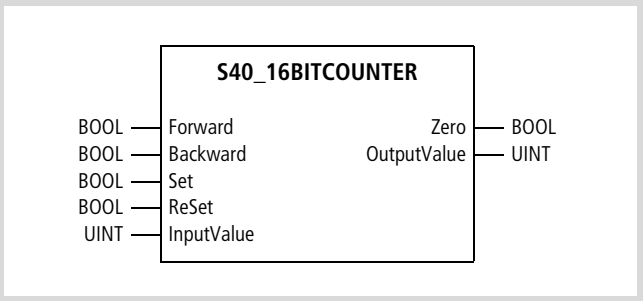
Description
The function block "Real_To_IEEE" converts a number of data type REAL to a floating point number in the IEEE-754 single-precision format. The representation of the float number requires four bytes (32 bits). This conversion changes the format only, not the value:

EXPO_Out	Mant_2_Out	Mant_1_Out	Mant_0_Out

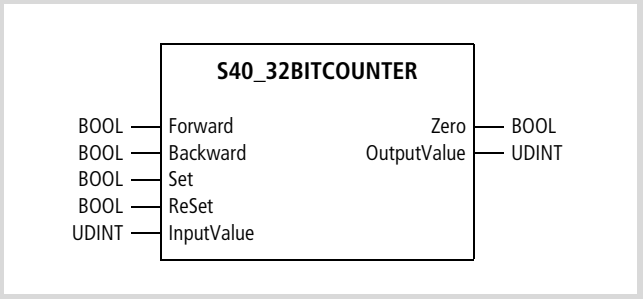
The IEEE format is mainly used for transferring floating point numbers to other systems (e.g. via PROFIBUS).

CounterFunctionBlocks

S40_16BitCounter
S40_32BitCounter
Up/down counter



Function block prototype



Function block prototype

Meanings of the operands

Forward	Forward counting pulse, rising edge
Backward	Reverse counting pulse, rising edge
Set	Setting condition, rising edge
ReSet	Reset condition
InputValue	Input value
Zero	Signal: count = 0
OutputValue	Count

Description

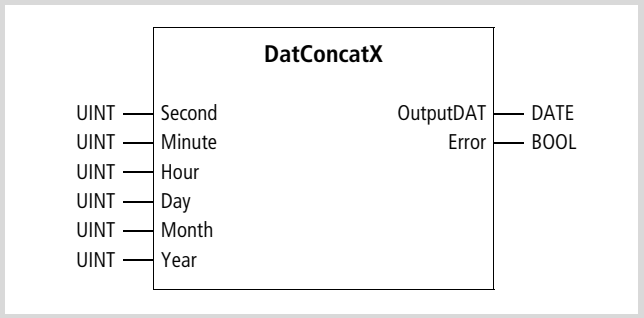
This function block provides a forward and backward count. Each rising edge at input operand "Forward" increments the count, i.e. increases it by "1". A rising edge of "Backward" decreases the count by "1". The current count is indicated by operand "OutputValue". With a rising edge at "Set", the value of "InputValue" is written to the counter. State "1" of "ReSet" clears the count. While a "1" is applied, the remaining count functions are inhibited.

The count ranges from 0 to 65,535 (16-bit) or 4,294,967,295 (32-bit). When the count reaches 65535 or 4294967295 respectively, it returns to zero at the next counting pulse. If zero is reached during reverse counting, the next counting pulse sets the value to 65,535 or 4,294,967,295. A carry is not formed in either case.

If the counter is implemented as a non-retentive instance, it is initialized at "0", output operand "Zero" indicating a value of "1".
Counters implemented as retentive instances keep their original values.

Date and Time

DATconcatX
generate date and time



Function block prototype

Meanings of the operands

Second	Seconds element of date and time
Minute	Minutes element of date and time
Hour	Hours element of date and time
Day	Days element of date and time
Month	Months element of date and time
Year	Years element of date and time
OutputDAT	Resulting value for date and time
Error	Error message

Description

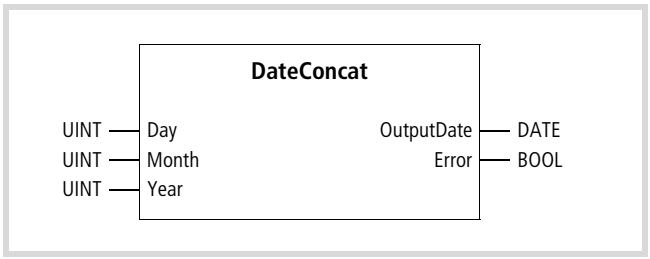
The values that are given as input operands for the data type UINT are combined to form a value for date and time in the output operand "OutputDAT".

The function block processes the individual values of the input operands into the corresponding elements for date and time. The input operands must not go outside the specific value ranges for a dat or a time. The year number must lie within the value range 1993 to 2092. If a given value range is infringed, then the "1" status at the "Error" output indicates an error.

Example: "Generate date and time"

```
PROGRAM dtgen
VAR
  DateTimeConcatenation : DATconcatX;
  Second : UINT := 30;
  Minute : UINT := 10;
  Hour : UINT := 3;
  Day : UINT := 12;
  Month : UINT := 12;
  Year : UINT := 1997;
  Error : BOOL;
  Date_1 : DATE_AND_TIME;
END_VAR
  CAL DateTimeConcatenation   (Second := Second,
                               Minute := Minute,
                               Hour := Hour,
                               Day := Day,
                               Month := Month,
                               Year := Year,
                               OutputDAT=>Date_1,
                               Error=>Error)
END_PROGRAM
```

DateConcat
generate date



Function block prototype

Meanings of the operands

Day	Day element of a date
Month	Month element of a date
Year	Year element of a date
OutputDate	Resulting value for the date
Error	Error message

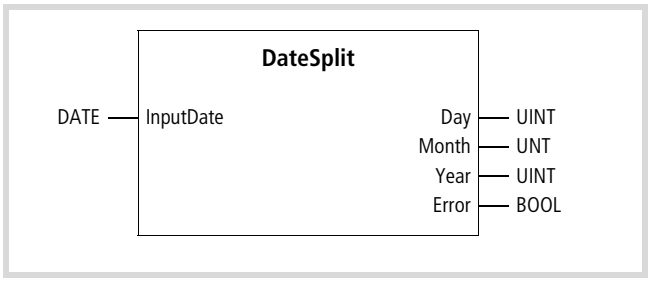
Description

The values that are given as input operands for the data type UINT are combined to form a value for a date in the output operand "OutputDate". The input operands must not go outside the specific value ranges for a date, the year number must lie within the range 1993 to 2092. If the given value ranges are infringed, then the "1" status at the "Error" output indicates an error.

Example: "Generate date"

```
PROGRAM datumgen
VAR
  Date concatenation :
  DateConcat;
  Date : Date;
  Day : UINT := 24;
  Month : UINT := 12;
  Year : UINT := 1996;
  Error : BOOL;
END_VAR
  CAL Date concatenation          (Day := Day,
                                  Month := Month,
                                  Year := Year,
                                  OutputDate=>Date,
                                  Error=>Error)
END_PROGRAM
```

DateSplit
Split data type DATE



Function block prototype

Meanings of the operands

InputDate	Date
Day	Day element of date
Month	Month element of date
Year	Year element of date
Error	Error message

Description

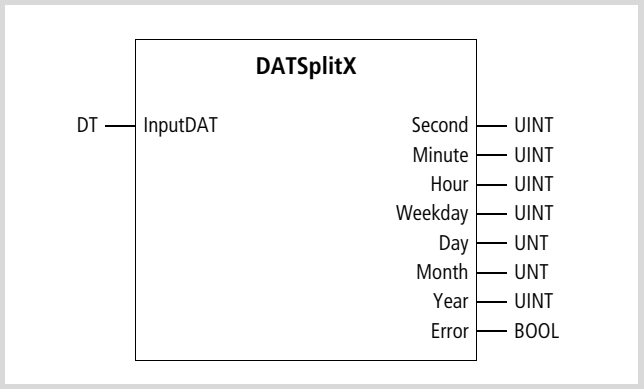
The date value given by the input operand "InputDate" is split into its elements, which are then each given out as data type UINT.

The year number must lie within the range 1993 to 2092. If an invalid year number is entered, then the output operand "Error" has status "1" to indicate an error.

Example: "Split date value"

```
PROGRAM datumtln
VAR
  DateElements : DateSplit;
  Date : Date := D#1999-12-24;
  Day : UINT;
  Month : UINT;
  Year : UINT;
  Error : BOOL;
END_VAR
  CAL DateElements              (InputDate := Date,
                                Day=>Day,
                                Month=>Month,
                                Year=>Year,
                                Error=>Error)
END_PROGRAM
```

DATSplitX
Split data type DATE_AND_TIME (DT)



Function block prototype

Meanings of the operands

InputDAT	Date and time
Second	Seconds element of date and time
Minute	Minutes element of date and time
Hour	Hours element of date and time
Weekday	Weekday (0 = Sunday)
Day	Days element of date and time
Month	Months element of date and time
Year	Years element of date and time
Error	Error message

Description

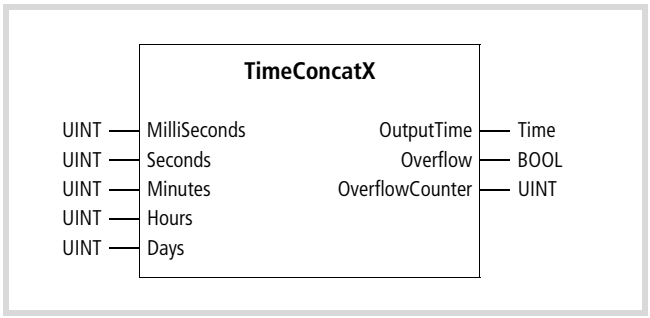
The date and time value given by the input operand "InputDAT" is split into its elements, which are then each given out as data type UINT.

The year number must lie within the range 1993 to 2092. If an invalid year number is entered, then the output operand "Error" has status "1" to indicate an error.

Example: "Split date and time"

```
PROGRAM dtspalt
VAR
 DT_Elements : DATsplitX;
 Date_1 : DT := DT#1999-11-20-22:13:12;
 Second : UINT;
 Minute : UINT;
 Hour : UINT;
 Weekday: UINT;
 Day : UINT;
 Month : UINT;
 Year : UINT;
 Error : BOOL;
END_VAR
 CAL DT_Elements                (InputDAT := Date_1,
                                 Second=>second,
                                 Minute=>minute,
                                 Hour=>hour,
                                 Weekday=>weekday,
                                 Day=>day,
                                 Month=>month,
                                 Year=>year,
                                 Error=>error)
END_PROGRAM
```

TimeConcatX
generate time period



Function block prototype

Meanings of the operands

MilliSeconds	Millisecond element of a time period
Seconds	Second element of a time period
Minutes	Minute element of a time period
Hours	Hour element of a time period
Days	Day element of a time period
OutputTime	Resulting value for a time period
Overflow	Overflow display
OverflowCounter	Number of overflows

Description

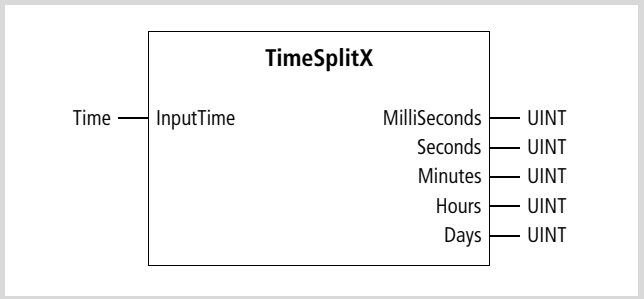
The individual time elements that are given as input operands are combined to form a value for a time period of data type TIME in the output operand "OutputTime".

The function block processes the individual values into the corresponding elements for the time period, and produces a carry value if necessary. The input variables can have any permissible value for the data type UINT, as long as the total result is not larger than T#49d17h2m47s295ms (= $2^{32} - 1$). If this value is exceeded, then the "1" status appears at the "Error" output, indicating an error. The number of overflows is shown by the "OverflowCounter" output.

Example: "Generate time period from elements"

```
PROGRAM zeit_gen
VAR
  TimeConcatenation : TimeConcatX;
  Milliseconds : UINT;
  Seconds : UINT;
  Minutes : UINT;
  Hours : UINT;
  Days : UINT;
  TimeStructure : TIME;
  Overflow : BOOL;
  Overflow counter : UINT;
END_VAR
  CAL TimeConcatenation  (MilliSeconds := Milliseconds,
                          Seconds := Seconds,
                          Minutes := Minutes,
                          Hours := Hours,
                          Days := Days,
                          OutputTime=>TimeStructure,
                          Overflow=>Overflow,
                          OverflowCounter=>Overflow counter)
END_PROGRAM
```

TimeSplitX
Split variable type TIME



Function block prototype

Meanings of the operands

InputTime	Time period to be split into time elements
MilliSeconds	Milliseconds element of the time period
Seconds	Seconds element of the time period
Minutes	Minutes element of the time period
Hours	Hours element of the time period
Days	Days element of the time period

Description

The time period given by the input operand "InputTime" is split into its time elements, which are then each given out as data type UINT.

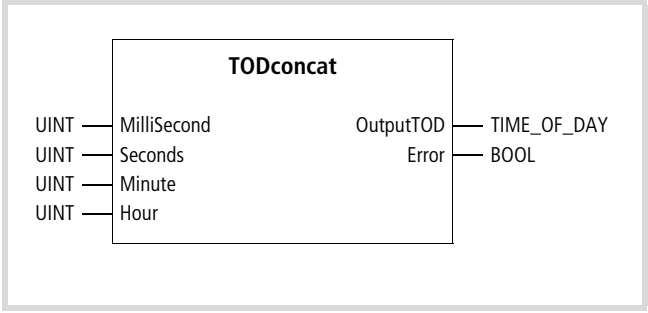
The maximum permissible time period is T#49d17h2m47s295ms (= $2^{32} - 1$).

Important
If the range is exceeded, data type TIME will be altered to the residual value, without an error message.

Example: "Split time period into time elements"

```
PROGRAM zt_spalt
VAR
  TimeElements : TimeSplitX;
  Period : TIME := T#49d17h2m47s295ms;
  Milliseconds : UINT;
  Seconds : UINT;
  Minutes : UINT;
  Hours : UINT;
  Days : UINT;
END_VAR
  CAL TimeElements          (InputTime := Period,
                             MilliSeconds=>Milliseconds,
                             Seconds=>Seconds,
                             Minutes=>Minutes,
                             Hours=>Hours,
                             Days=>Days)
END_PROGRAM
```

TODconcat
Generate time of day



Function block prototype

Meanings of the operands

MilliSecond	Milliseconds element of a time of day
Second	Seconds element of a time of day
Minute	Minutes element of a time of day
Hour	Hours element of a time of day
OutputTOD	Resulting value for the time of day
Error	Error message

Description

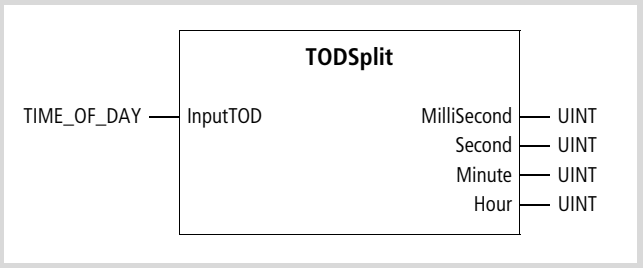
The individual time elements that are given as input operands are combined to form a value for the time of day in the output operand "OutputTOD". The input operands must not go outside the specific value ranges for a time of day.

If the permissible range for an input operand is infringed, then the "1" status appears at the "Error" output to indicate an error.

Example: "Generate time of day from elements"

```
PROGRAM time2
VAR
  TimeOfDay : TODconcat;
  Milliseconds : UINT;
  Second : UINT;
  Minute : UINT;
  Hour : UINT;
  DayTimeStructure : TIME_OF_DAY;
  Error : BOOL;
END_VAR
  CAL TimeOfDay             (MilliSecond := Milliseconds,
                             Second := Second,
                             Minute := Minute,
                             Hour := Hour,
                             OutputTOD=>DayTimeStructure,
                             Error=>Error)
END_PROGRAM
```

TODSplit
Split data type TIME_OF_DAY



Function block prototype

Meanings of the operands

InputTOD	Time of day
MilliSecond	Milliseconds element of the time of day
Second	Seconds element of the time of day
Minute	Minutes element of the time of day
Hour	Hours element of the time of day

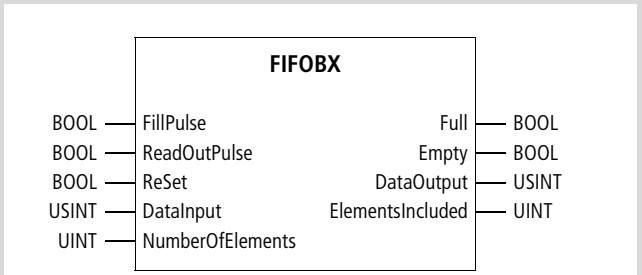
Description
The time value given by the input operand "InputTOD" is split into its time elements, which are then each given out as data type UINT.

Example: "Split time of day into time elements"

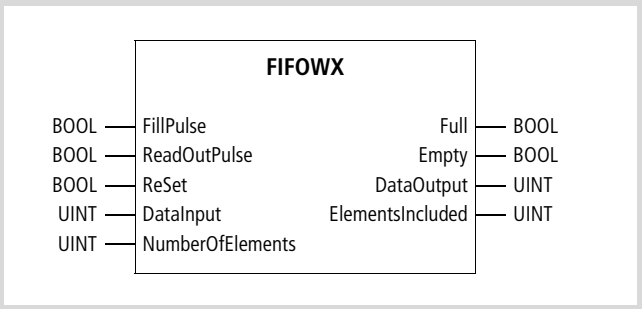
```
PROGRAM time1
VAR
  TimeOfDay_split : TODsplit;
  TimeOfDay : TOD := TOD#12:13:12;
  Milliseconds : UINT;
  Seconds : UINT;
  Minutes : UINT;
  Hours : UINT;
END_VAR
  CAL TOD_split            (InputTOD := TimeOfDay,
                           MilliSecond=>Milliseconds,
                           Second=>Seconds,
                           Minute=>Minutes,
                           Hour=>Hours)
END_PROGRAM
```

RegisterFunctionBlocks

FifoBx
FifoWx
FIFO register (8/16-bit)



Function block prototype



Function block prototype

Meanings of the operands

FillPulse	Fill pulse
ReadOutPulse	Read-out pulse
ReSet	Reset
DataInput	Data input
Number Of Elements	Length of register (1 to 128)
Full	Register full
Empty	Register empty
DataOutput	Data output
ElementsIncluded	Elements in the register

Description
The register length specifies the number of register fields.

When operand "FillPulse" has a rising edge, the value of operand "DataInput" is written to the deepest free register field. The register can be filled with every "FillPulse" until all register fields are filled. The register's "Full" state is indicated by operand "Full" = "1".

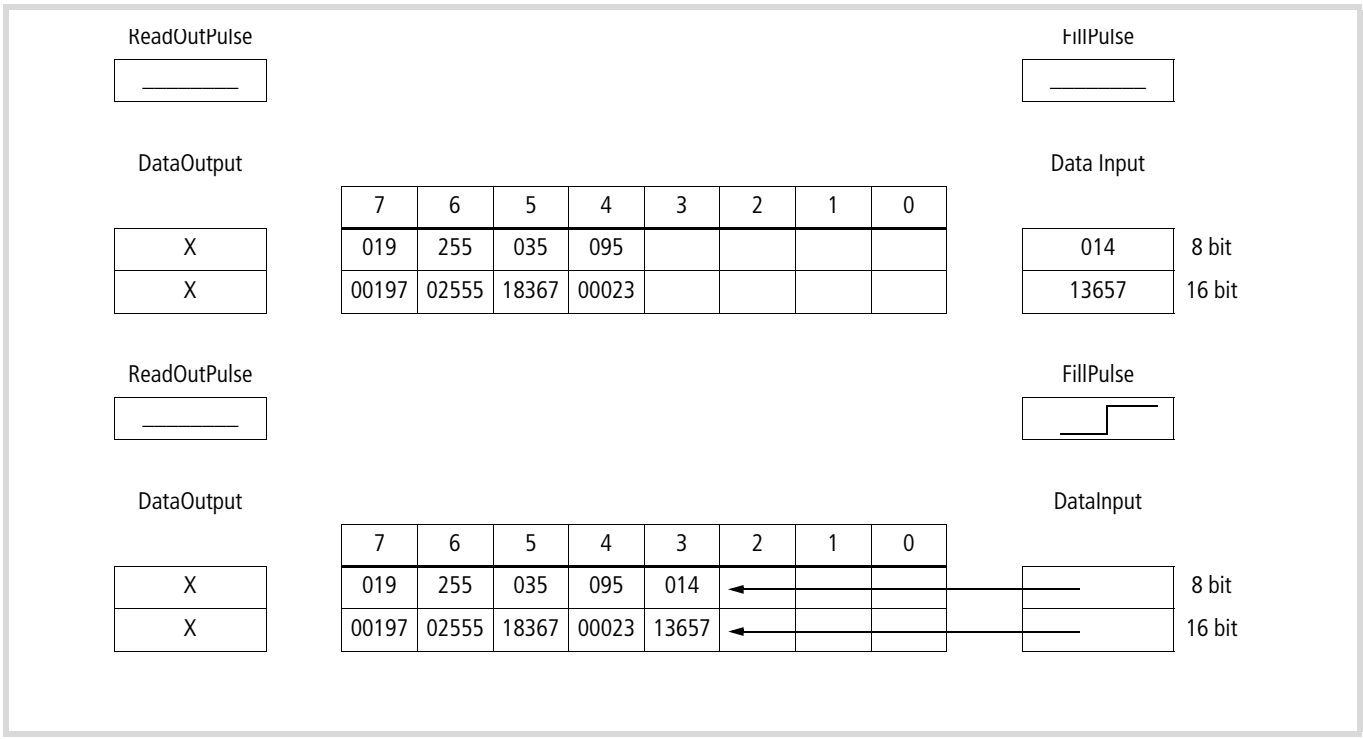


Figure 2: Content of the queue before and after a fill pulse

With a rising edge of operand "ReadOutPulse", the first written register field – which is at the deepest position in the queue – is read and passed to operand "DataOutput". Every further pulse of "ReadOutPulse" results in further elements in the register being passed to operand "DataOutput". When the last register field has been read, "Empty" changes to state "1" to indicate that the Register is empty.

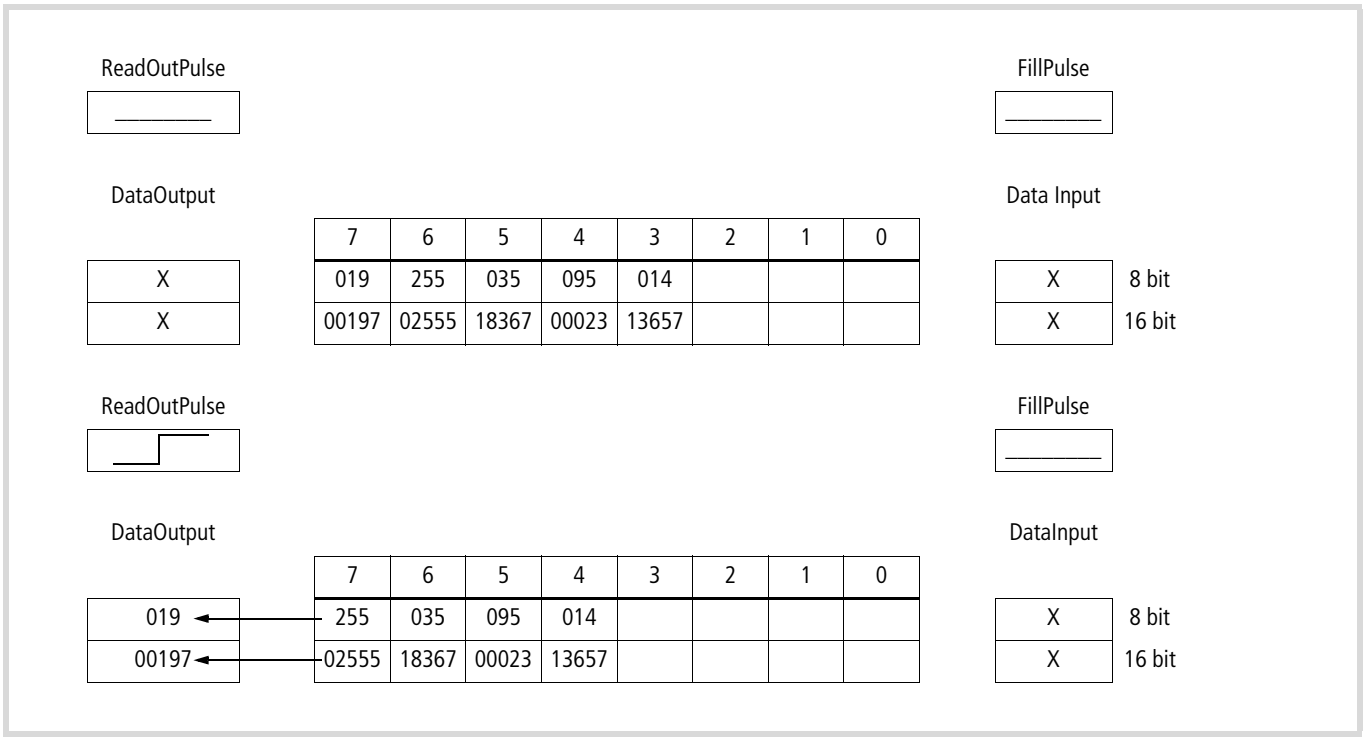
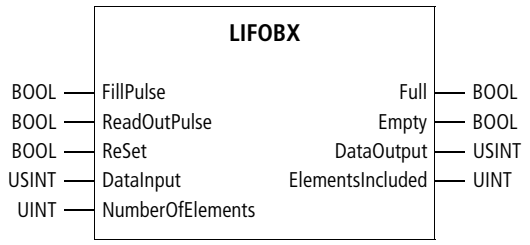


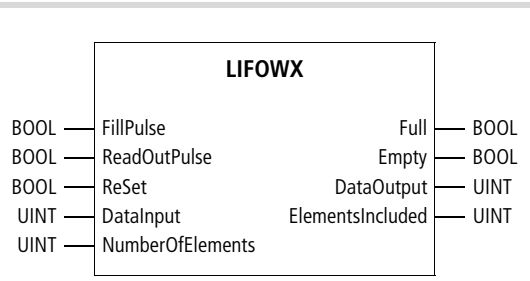
Figure 3: Queue content before and after a read-out pulse

State "1" of operand "ReSet" clears the entire register.

LifoBx
LifoWx
LIFO register (8/16-bit)



Function block prototype



Function block prototype

Meanings of the operands

FillPulse	Fill pulse
ReadOutPulse	Read-out pulse
ReSet	Reset
DataInput	Data input
Number Of Elements	Length of register (1 to 128)
Full	Register full
Empty	Register empty
DataOutput	Data output
ElementsIncluded	Elements in the register

Description

The register length specifies the number of register fields.

When operand "FillPulse" has a rising edge, the value of operand "DataInput" is written to the deepest free field of the stack register. The stack register can be filled with every "FillPulse" until all register fields are filled. The stack register's "Full" state is indicated by operand "Full" = "1".

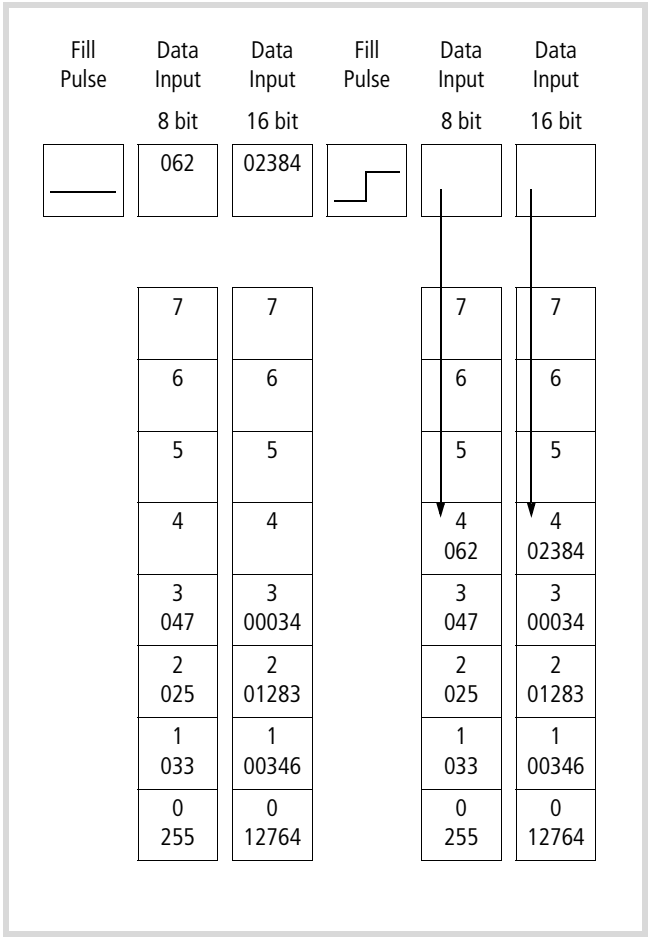


Figure 4: Content of the stack register before and after a fill pulse

With a rising edge of operand "ReadOutPulse", the last written register field is read and passed to operand "DataOutput". Every further pulse of "ReadOutPulse" results in further elements in the register being passed to operand "DataOutput". When the last register field has been read, "Empty" changes to state "1" to indicate that the Register is empty.

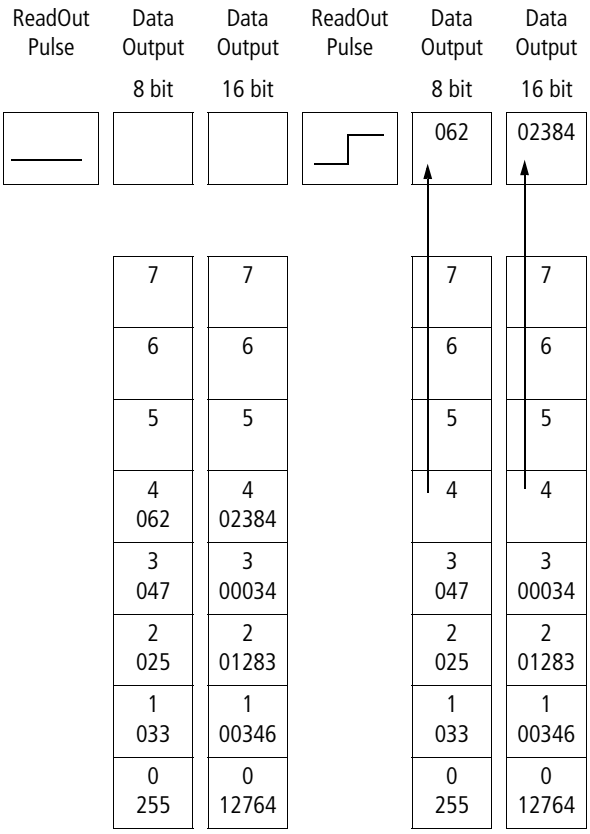
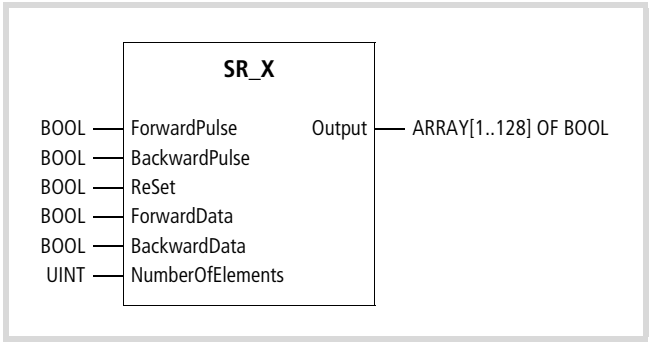


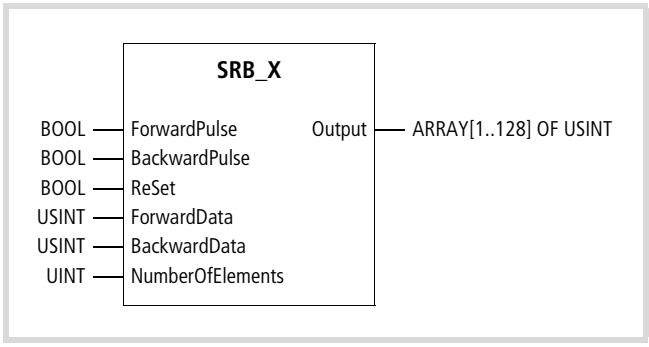
Figure 5: Content of the stack register before and after a read-out pulse

State "1" of operand "ReSet" clears the entire stack register.

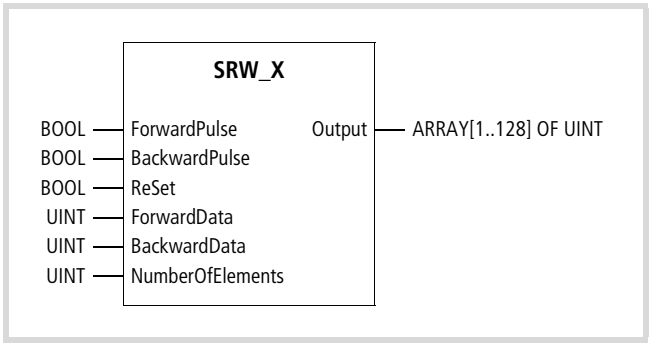
SR_x
SRB_x
SRW_x
Shift register



Function block prototype



Function block prototype



Function block prototype

Meanings of the operands

ForwardPulse	Pulse input, forward
BackwardPulse	Pulse input, reverse
ReSet	Reset
ForwardData	Data input, forward
BackwardData	Data input, reverse
Number Of Elements	Length of register (1 to 128)
Output	Output (ARRAY 1 ... 128)

Description

The register length specifies the number of output register fields.

When operand "ForwardPulse" has a rising edge, the value of operand "ForwardData" is written to the first element "Output [1]". The original contents are shifted up by one element.

When operand "BackwardPulse" has a rising edge, the value of operand "BackwardData" is written to the last element. The original contents are shifted down by one element.

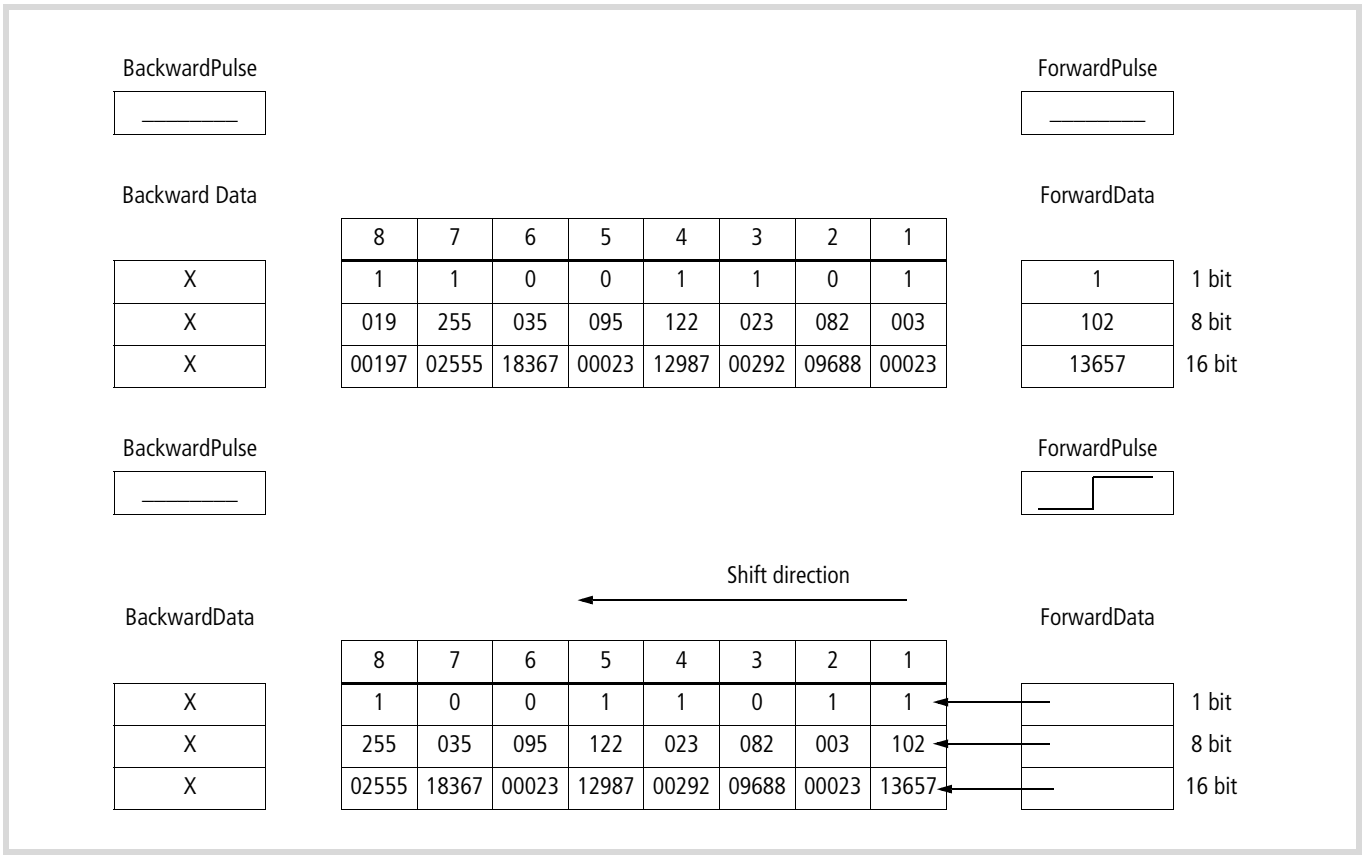


Figure 6: Content of the shift register before and after a forward pulse

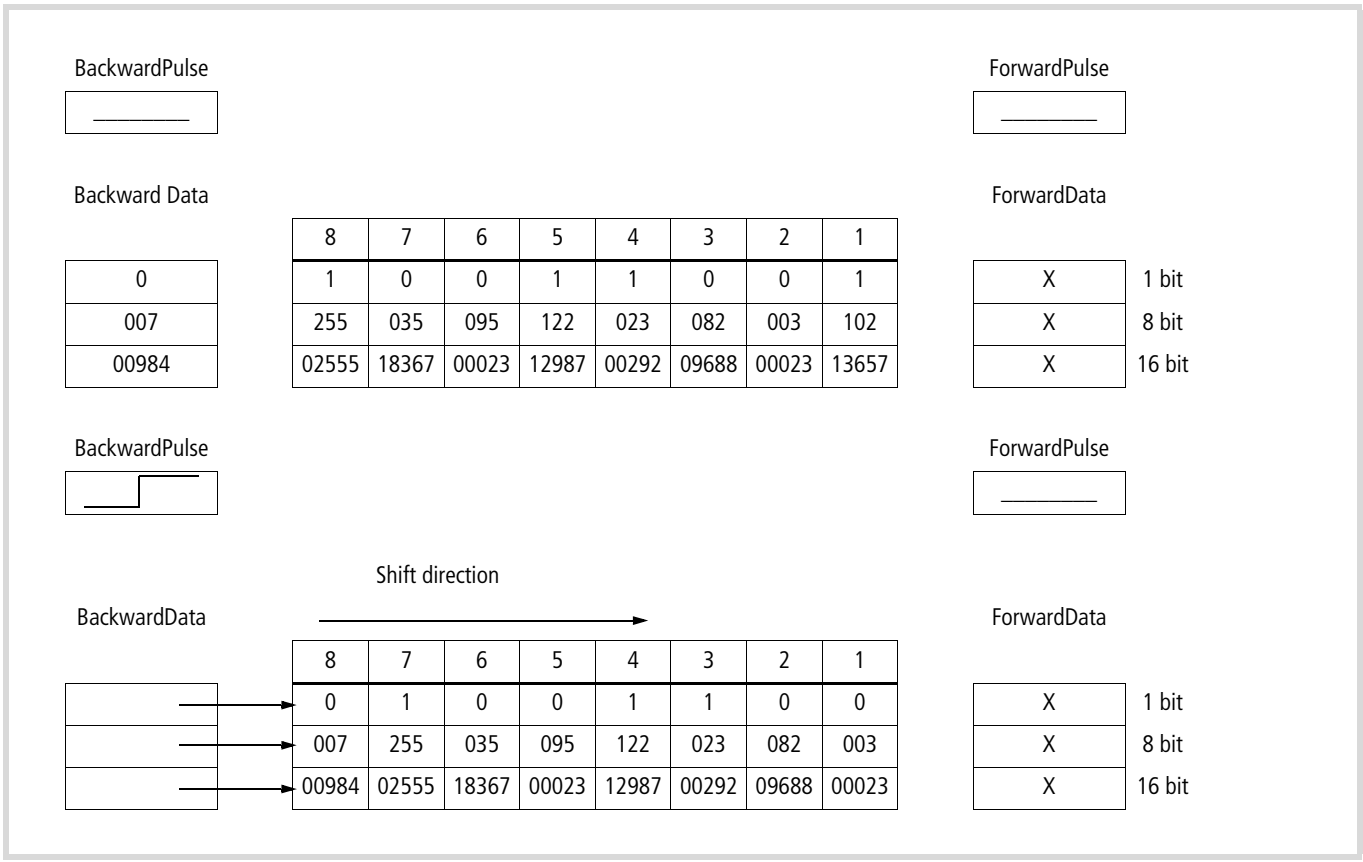
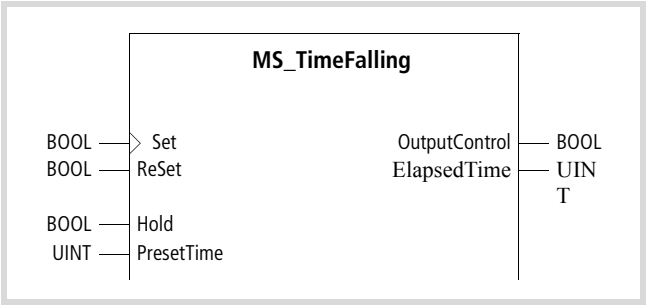


Figure 7: Content of the shift register before and after a reverse pulse

State "1" of operand "ReSet" clears the entire array.

Timer

MS_TimeFalling
Switch-off delay timer, milliseconds



Function block prototype

Meanings of the operands

Set	Start condition, rising edge
ReSet	Reset condition
Hold	Timer interrupt
PresetTime	Preset time in milliseconds
OutputControl	Control output
ElapsedTime	Elapsed time in milliseconds

Description

A rising edge on the "Set" input loads the "PresetTime" value into the timer, as the delay time "T" in milliseconds. The output "OutputControl" goes to the "1" state. The timer starts when the "Set" input switches back to "0". The "OutputControl" output switches off after the delay time "T", i.e. it returns to the "0" state (1). The "ElapsedTime" output indicates the current time value in milliseconds. If the "0" state at the "Set" input is only present for a shorter time than "T", then the "Output-Control" output remains at "1".

The running of the initiated timer can be interrupted by a "1" at the "Hold" input. The interrupted time period will continue if the "Hold" input returns to "0". The switch-off delay time is then lengthened by the duration of the "Hold" signal, and is thus "$T+T_{HALT}$" (2).

If a "1" is present at the "Hold" input when the rising edge appears at the "Set" input, then the loading of the "PresetTime" value is delayed as long as the "1" is present at the "Hold" input. The appearance of the "1" state at the "OutputControl" output will also be delayed accordingly (4).

If the state of the "Set" input changes from "1" to "0" while a "1" is present at the "Hold" input, then the timer will start when the "Hold" input returns to the "0" state. The switch-off delay time is then lengthened by the duration of the Hold" signal, and is thus "$T+T_{HALT}$" (3).

If the "Set" input switches to the "1" state and then back to "0", while the "Hold" input remains at "1", then the "OutputControl" output will remain at "0".

If a "1" is present at the "ReSet" input, the timer will be reset. A rising edge at the "Set" input will only start the timer when the "ReSet" state changes back from "1" to "0" (5).

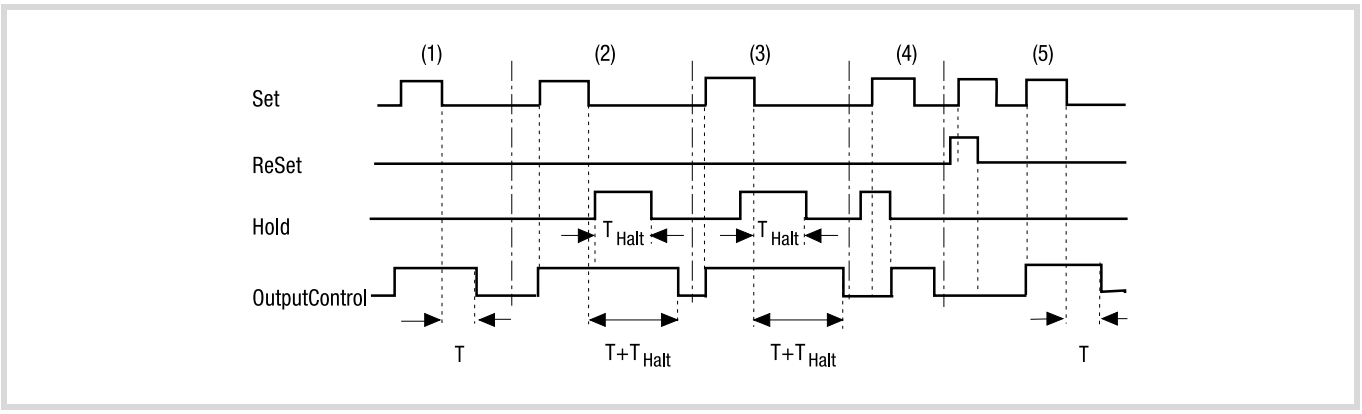
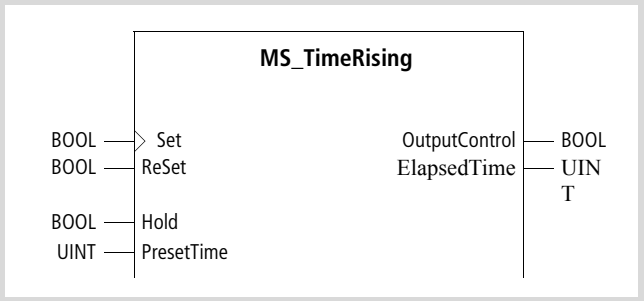


Figure 8: Timing diagram

Example: "Switch-off delay, 25 milliseconds"

```
PROGRAM time4
VAR
 Timer4 : S_TimeFalling;
 Start : BOOL;
 Halt : BOOL;
 TimerValue4 : UINT := 25; (* TimerValue4 = 25 milliseconds *)
 Output4 : BOOL;
END_VAR
 CAL Timer4              (Set := Start,
                          Hold := Hold,
                          PresetTime := TimerValue4)
 LD Timer4.OutputControl
 ST Output4
END_PROGRAM
```

MS_TimeRising
Switch-on delay timer, milliseconds



Function block prototype

Operand significance

Set	Start condition, rising edge
ReSet	Reset condition
Hold	Timer interrupt
PresetTime	Preset time in milliseconds
OutputControl	Control output
ElapsedTime	Elapsed time in milliseconds

Description

A rising edge on the "Set" input loads the "PresetTime" value into the timer, as the time "T" in milliseconds, and the timer starts. The "Output control" output switches to the "1" state after the preset delay time T", and remains in this state until the "Set" input returns to the "0" state (1). The "ElapsedTime" output indicates the current time value in milliseconds. If the "1" state at the "Set" input is only present for a shorter time than "T", then the "Output-Control" output remains at "0".

The running of the initiated timer can be interrupted by a "1" at the "Hold" input. The interrupted time period will continue if the "Hold" input returns to "0". The delay time is then lengthened by the duration of the "Hold" signal, and is thus "$T+T_{HALT}$" (2).

If a "1" is present at the "Hold" input when the rising edge appears at the "Set" input, then the start procedure is delayed as long as the "1" is present at the "Hold" input (4). If the state of the "Set" input changes from "1" to "0" after the preset time has elapsed, and while a "1" is present at the "Hold" input, then "OutputControl" will switch to "0" when "Hold" returns to the "0" state (3).

If a "1" is present at the "ReSet" input, the timer will be reset. It is not possible to start the timer in this condition. A rising edge at the "Set" input will only be effective as a timer start after "ReSet" state changes back from the "1" to the "0" state (5).

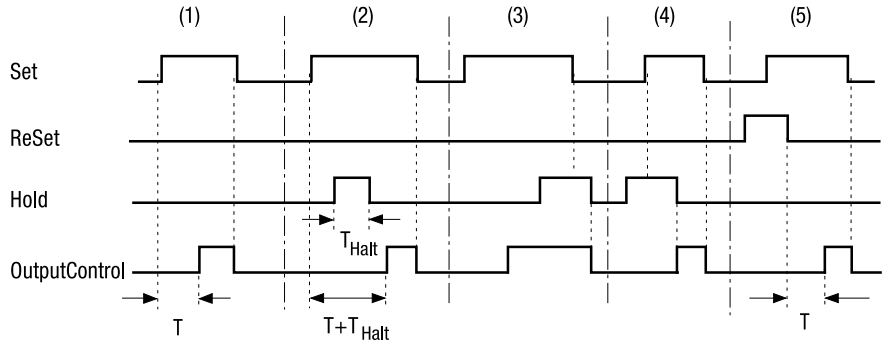


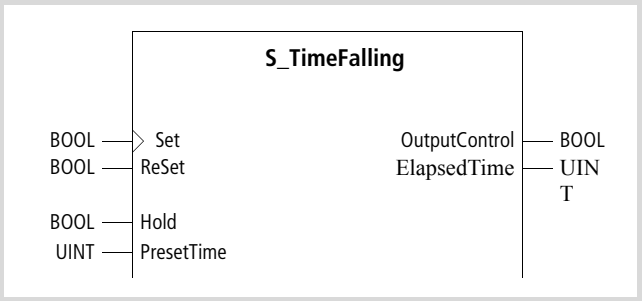
Figure 9: Timing diagram

Example: "Switch-on delay, 7 milliseconds"

```
PROGRAM time3
VAR
 Timer3 : MS_TimeRising;
 Start : BOOL;
 Halt : BOOL;
 Reset : BOOL;
 TimeValue3: UINT := 7;    (* TimeValue3 = 7 milliseconds *)

 ElapsedTime3 : BOOL;
 Output3 : BOOL;
END_VAR
```

```
 CAL Timer3(Set := Start,
                          ReSet := Reset,
                          Hold := Hold,
                          PresetTime := TimeValue3)
 LD Timer3.OutputControl
 ST Output3
 LD Timer3.ElapsedTime
 ST ElapsedTime3
END_PROGRAM
```

S_TimeFalling
Switch-off delay timer, seconds



Function block prototype

Meanings of the operands

Set	Start condition, rising edge
ReSet	Reset condition
Hold	Timer interrupt
PresetTime	Preset time in seconds
OutputControl	Control output
ElapsedTime	Actually elapsed time in seconds

Description

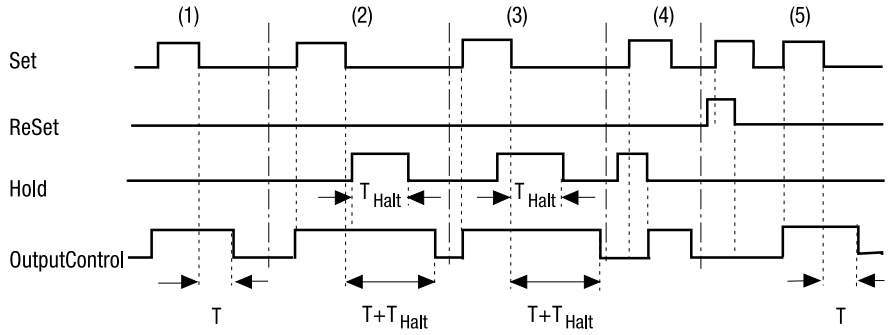


Figure 10: Timing diagram

A rising edge on the "Set" input loads the "PresetTime" value into the timer, as the delay time "T" in seconds. The output "OutputControl" goes to the "1" state. The timer starts when the "Set" input switches back to "0". The "OutputControl" output switches off after the delay time "T", i.e. it returns to the "0" state (1). The "ElapsedTime" output indicates the actually elapsed time, in milliseconds. If the "0" state at the "Set" input is only present for a shorter time than "T", then the "Output-Control" output remains at "1".

The running of the initiated timer can be interrupted by a "1" at the "Hold" input. The interrupted time period will continue if the "Hold" input returns to "0". The switch-off delay time is then lengthened by the duration of the "Hold" signal, and is thus "$T+T_{HALT}$" (2).

If a "1" is present at the "Hold" input when the rising edge appears at the "Set" input, then the loading of the "PresetTime" value is delayed as long as the "1" is present at the "Hold" input. The appearance of the "1" state at the "OutputControl" output will also be delayed accordingly (4).

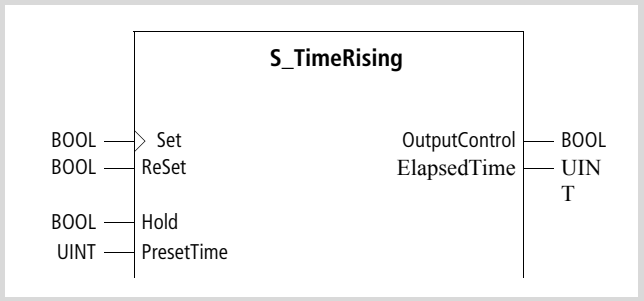
If the state of the "Set" input changes from "1" to "0" while a "1" is present at the "Hold" input, then the timer will start when the "Hold" input returns to the "0" state. The switch-off delay time is then lengthened by the duration of the Hold" signal, and is thus "$T+T_{HALT}$" (3).

If a "1" is present at the "ReSet" input, the timer will be reset. A rising edge at the "Set" input will only be effective as a timer start after "ReSet" state changes back from the "1" to the "0" state.

Example: "Switch-off delay, 32 seconds"

```
PROGRAM time2
VAR
 Timer2 : S_TimeFalling;
 Start : BOOL;
 Halt : BOOL;
 TimeValue2 : UINT:= 32; (* TimeValue2 = 32 seconds *)
 Output2 : BOOL;
 ElapsedTime2 : UINT;
END_VAR
 CAL Timer2              (Set := Start,
                         Hold := Hold,
                         PresetTime := TimeValue2)
 LD Timer2.OutputControl
 ST Output2
 LD Timer2.ElapsedTime
 ST ElapsedTime2
 ...
```

S_TimeRising
Switch-on delay timer, seconds



Function block prototype

Meanings of the operands

Set	Start condition, rising edge
ReSet	Reset condition
Hold	Timer interrupt
PresetTime	Preset time in seconds
OutputControl	Control output
ElapsedTime	Actually elapsed time in seconds

Description

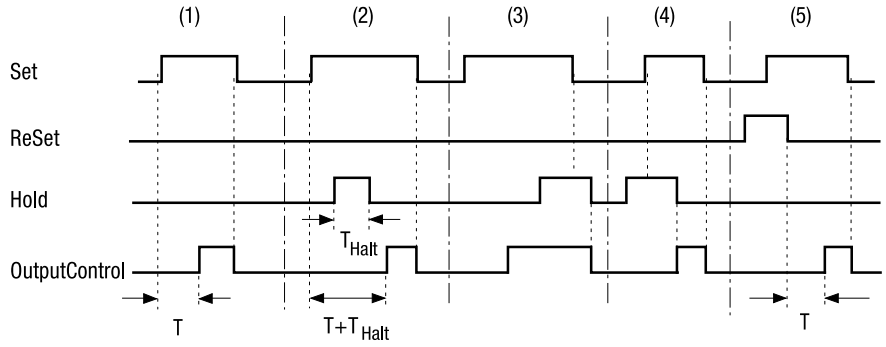


Figure 11: Timing diagram

A rising edge on the "Set" input loads the "PresetTime" value into the timer, as the delay time "T" in seconds, and the timer starts. The "Output control" output switches to the "1" state after the preset delay time T", and remains in this state until the "Set" input returns to the "0" state (1). The "ElapsedTime" output indicates the actually elapsed time, in milliseconds. If the "1" state at the "Set" input is only present for a shorter time than "T", then the "Output-Control" output remains at "0".

The running of the initiated timer can be interrupted by a "1" at the "Hold" input. The interrupted time period will continue if the "Hold" input returns to "0". The switch-on delay time is then lengthened by the duration of the "Hold" signal, and is thus "$T+T_{HALT}$" (2).

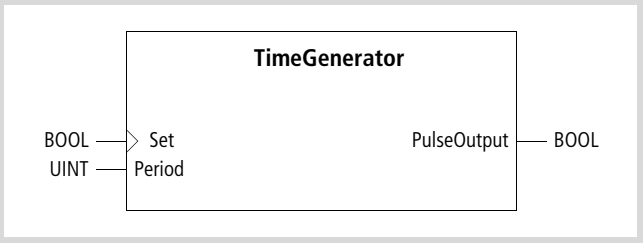
If a "1" is present at the "Hold" input when the rising edge appears at the "Set" input, then the start procedure is delayed as long as the "1" is present at the "Hold" input (4). If the state of the "Set" input changes from "1" to "0" after the preset time has elapsed, and while a "1" is present at the "Hold" input, then "OutputControl" will switch to "0" when "Hold" returns to the "0" state (3).

If a "1" is present at the "ReSet" input, the timer will be reset. It is not possible to start the timer in this condition. A rising edge at the "Set" input will only start the timer when the "ReSet" state changes back from "1" to "0" (5).

Example: "Switch-on delay, 12 seconds"

```
PROGRAM time1
VAR
 Timer1 : S_TimeRising;
 Start : BOOL;
 Halt : BOOL;
 TimeValue1 : UINT := 12; (* TimeValue1 = 12 seconds *)
 Output1 : BOOL;
END_VAR
 CAL Timer1               (Set := Start,
                          Hold := Hold,
                          PresetTime := TimeValue1)
 LD Timer1.OutputControl
 ST Output1
...
```

TimeGenerator
Clock generator



Function block prototype

Meanings of the operands

Set	Start condition, rising edge
Period	Pulse repetition period in milliseconds
PulseOutput	Pulse output

Description

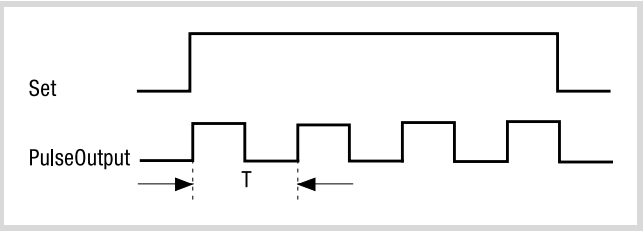


Figure 12: Timing diagram

A rising edge on the "Set" input loads the "Period" value into the timer, as the period repetition time "T" in milliseconds. As long as a "1" is present at the "Set" input, the output "PulseOutput" will produce a pulse train with the preset pulse repetition period and a 1:1 duty cycle.

The period can be set from 1 to 65535 milliseconds at the "Period" input.

If the "Period" value is altered while the clock generator is running, it will not have an immediate effect on the frequency. The new value only becomes effective with the next rising edge at the "Set" input.

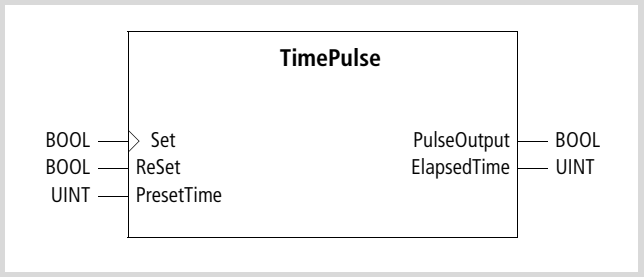
The accuracy of the output period is influenced by the cycle time of the program and the processing time of the operating system, and can be falsified by these amounts.

An odd value for the period will be rounded down to the nearest even value. If the period has a length that is of a similar magnitude to the cycle time, or even shorter, then the period of the output will be very imprecise. The error can be reduced by repeated calls of this function block within a program cycle.

Example "320 millisecond clock"

```
PROGRAM frequency
VAR
 ClockGenerator :
 TimeGenerator;
 Start : BOOL;
 PeriodTime: UINT := 320;
 Clock : BOOL;
END_VAR
 CAL ClockGenerator            (Set := Start,
                                Period := PeriodTime)
 LD ClockGenerator.PulseOutput
 ST Clock
...
```

TimePulse
Pulse timer



Function block prototype

Meanings of the operands

Set	Start condition, rising edge
ReSet	Reset condition
PresetTime	Preset time in milliseconds
PulseOutput	Control output
ElapsedTime	Elapsed time in milliseconds

Description

A rising edge on the "Set" input loads the "PresetTime" value into the timer, as the time "T" in milliseconds, and the timer starts. The output "PulseOutput" goes to the "1" state and remains in this state for the duration of the given time.

A change of the state of the "Set" input while the time is running has no effect on the sequence.

The "ElapsedTime" output indicates the current time value in milliseconds.

If a "1" is present at the "ReSet" input while the pulse time is running, the timer will be reset.

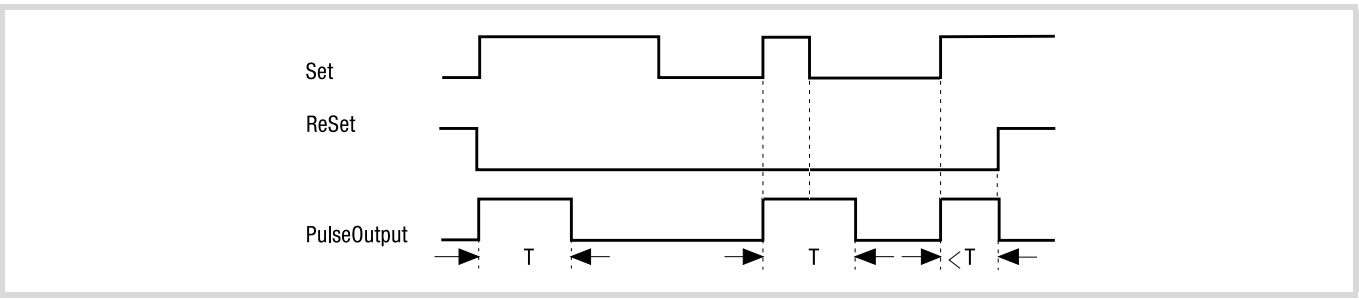


Figure 13: Timing diagram

Example: "125 millisecond pulse"

```
PROGRAM pulse
VAR
  Pause : TimePulse;
  Start : BOOL;
  PulseDuration: UINT := 125;
  OutputPulse : BOOL;
  ElapsedTIme : UINT;
END_VAR
  CAL Pause                   (Set := Start,
                              PresetTime := PulseDuration)
  LD Pause.PulseOutput
  ST OutputPulse
  LD Pause.ElapsedTime
  ST ElapsedTime
END_PROGRAM
```

2 Date and Time function blocks of the S40: XS40_MoellerFB_RTC.lib

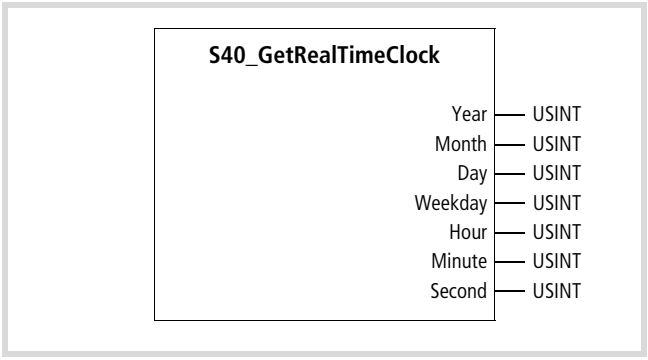
Library XS40_MoellerFB_RTC.lib contains the following function blocks:

- S40_GetRealTimeClock
- S40_Rtc
- S40_SetRealTimeClock

When you include this library, library SysLibRtc.lib is automatically included as well!

You can use the function blocks in the XC100/XC200.

S40_GetRealTimeClock
Read the real-time clock



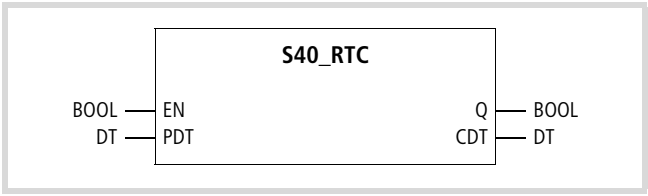
Function block prototype

Meanings of the operands

Year	current year number (2-figures)
Month	Month
Day	Day
Weekday	Weekday (0 = Sunday)
Hour	Hour
Minute	Minute
Second	Second

Description
If the function block "GetRealTimeClock" is called, it reads out the current time and date from the real-time clock and presents them as the parameters described above. An enable is not required, the block provides the values as soon as it is called.

S40_RTC
Set the real-time clock



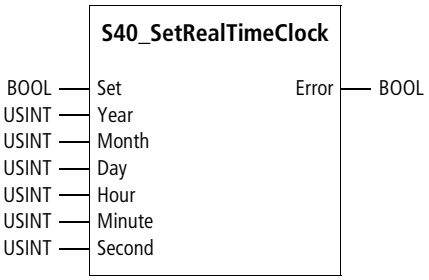
Function block prototype

Meanings of the operands

EN	Setting conditions
PDT	Real-time clock setting; valid years from 1993 to 2092
Q	Indicates valid value at CDT
CDT	Current date and time value

Description
The rising edge of operand EN the date and time settings specified by operand PDT are written to the real-time clock. Output operand Q indicates whether CDT is valid. The current time and date are output through operand CDT, regardless of the state of operand EN.

S40_SetRealTimeClock
Set the real-time clock



Function block prototype

Meanings of the operands

Set	Enable for accepting the applied values, L → H edge
Year	current year number (2-figures)
Month	Month
Day	Day
Hour	Hour
Minute	Minute
Second	Second
Error	Error message

Description
An L → H edge at input "Set" writes the input values.

3 Clock function blocks: RTCLib.lib

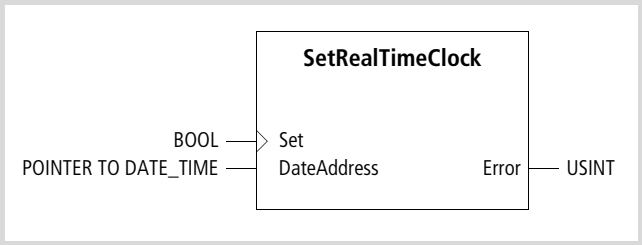
The library contains the following function blocks:

- SetRealTimeClock
- GetRealTimeClock

→ The function blocks of RTCLib.lib are only for use with the XC100.

Clock function blocks

SetRealTimeClock
Set the real-time clock



Function block prototype

Meanings of the operands

Set	Enable for accepting the given values, rising edge
POINTER TO DATE_TIME	Pointer to the address of the structure type DATE_TIME→ table 2
Error	Error messages

Table 2: Structure elements:

```
TYPE DATE_TIME:
STRUCT
  Second: USINT;
  Minute: USINT;
  Hour: USINT;
  Day: USINT;
  Month: USINT;
  Year: USINT;
  WeekDay: USINT;
END_STRUCT
END_TYPE
```

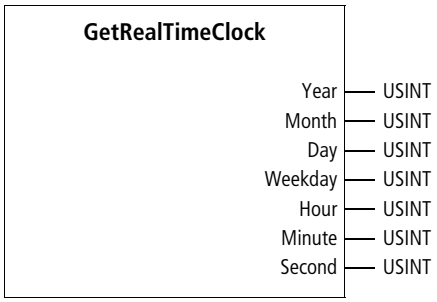
Description
A rising edge at the "Set" input sets the values for the real-time clock, which are then stored in a structure of type DATE_TIME. The set (given) values are contained in seven sequential data elements of type USINT. The byte contain, one after another: year, month, day, weekday, hour, minute and second.

A rising edge on the set input transfers all the values, but not partial information such as: the hour only, or the year only.

The "Error" output provides the error code:

1	Wrong parameter for DataAddress
2	Wrong year entered (valid range: 0 to 99)
3	Wrong month entered (valid range: 1 to 12)
4	Wrong day entered (valid range: 1 to 31)
5	Wrong weekday entered (valid range: 0 to 6, 0 = Sunday)
6	Wrong hour entered (valid range: 0 to 23)
7	Wrong minute entered (valid range: 0 to 59)
8	Wrong second entered (valid range: 0 to 59)

GetRealTimeClock
Read the real-time clock



Function block prototype

Meanings of the operands

Year	current year number (2-figures)
Month	Month
Day	Day
Weekday	Weekday (0 = Sunday)
Hour	Hour
Minute	Minute
Second	Second

Description

If the function block "GetRealTimeClock" is called, it reads out the current time and date from the real-time clock and presents them as the parameters described above. An enable is not required, the block provides the values as soon as it is called.

4 Visualisation function blocks: Visu.lib

The library contains the following function blocks (FB) and functions (FU):

- ClearLines (FB)
- ClearScreen (FB)
- EnableDisplay (FB)
- GetDisplayInfo (FB)
- GetTextAddress (FU)
- GetTextDBInfo (FB)
- InputValue (FB)
- SetBacklight (FB)
- SetContrast (FB)
- SetCursor (FB)
- WriteBargraph (FB)
- WriteLine (FB)
- WriteMultiString (FB)
- WriteMultiStringTextDB (FB)
- WriteMultiValue (FB)
- WriteString (FB)
- WriteStringTextDB (FB)
- WriteSysDate (FB)
- WriteSysDay (FB)
- WriteSysTime (FB)
- WriteValue (FB)

Visualisation blocks

General

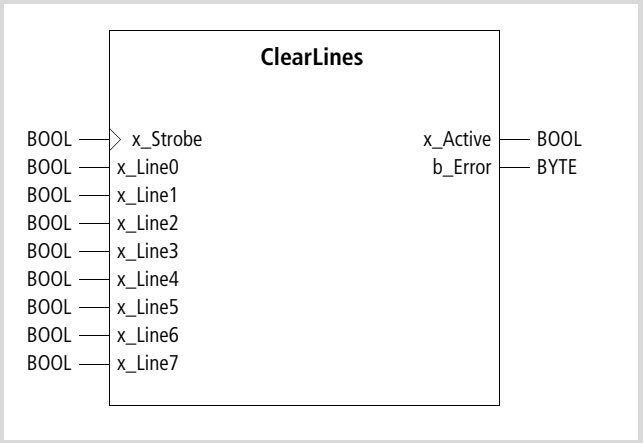
The visualisation blocks can be used not only for the XV-101-K42 (4 lines, each with 20 character) but also for the XV-101-K84 (8 lines, each with 40 characters).

The number of lines and characters given here applies when using a small-font character set (6 × 8 pixels). If you use a large-font character set (12 × 16 pixels), then both the number of lines and the number of characters per line are halved. The large display can then show 4 lines of 20 characters, and the small display can show 2 lines of 10 characters.

The "Blink" attribute can only be used with the large display.

A maximum of 10 function blocks can be active at the same time. If you call more than this, the error message 2 will appear "Over block number limit". The blocks will not be processed.

ClearLines
Delete the display lines



Function block prototype

Meanings of the operands

x_Strobe	A positive (rising) edge clears the parameter settings for the lines
x_Line0	Line number
...	Line number
x_Line7	Line number
x_Active	Block is active
b_Error	Error messages

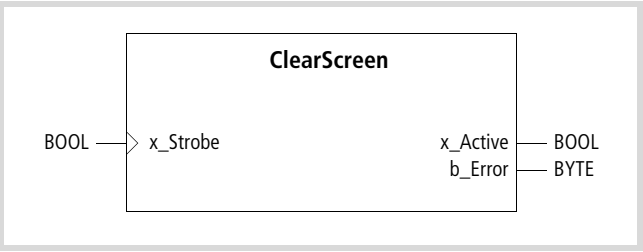
Description

A positive (rising) edge at the "x_Strobe" input means that the block deletes the line parameter settings for the inputs "x_Line0" to "x_Line7".

Error messages

0	No error
2	Over block number limit

ClearScreen
Delete the screen contents



Function block prototype

Meanings of the operands

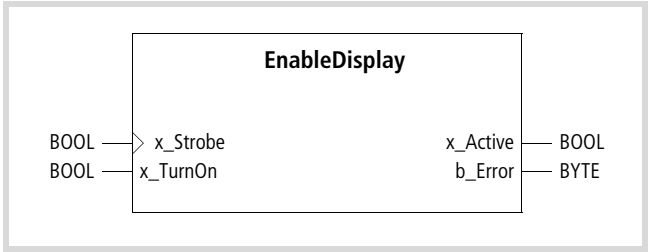
x_Strobe	A positive (rising) edge clears the display contents
x_Active	Block is active
b_Error	Error messages

Description
A positive (rising) edge at the "x_Strobe" input means that the block clears the display contents.

Error messages

0	No error
2	Over block number limit

EnableDisplay
Switch the display contents to visible/invisible



Function block prototype

Meanings of the operands

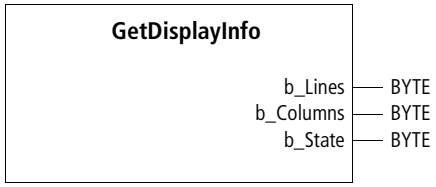
Strobe	The parameter "x_TurnOn" is accepted with a positive edge
TurnOn	Display contents visible = 1, display contents not visible = 0
Active	Block is active
Error	Error messages

Description
Depending on the state of the "x_TurnOn" input, a positive edge at the "x_Strobe" input makes the display contents visible/ invisible.

Error messages

0	No error
2	Over block number limit

GetDisplayInfo
Read the operating state and display type



Function block prototype

Meanings of the operands

b_Lines	Number of lines
b_Columns	Number of columns
b_State	MMI_STATE (operating state of the text display)

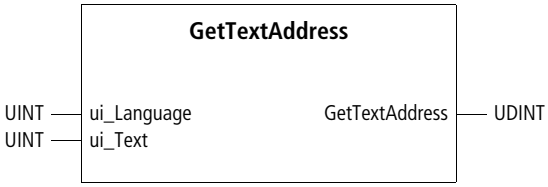
Description

The function block reads the present state of the text display and the type of display that is connected. When you call the function block, the number of characters per line for the connected text display is given out at the "b_Columns" output, and the number of lines at the "b_Lines" output. This applies for the small-font character set.

The "b_State" output shows the operating state of the display. The individual operating states are shown in data type MMI_STATE [ENUM]:

```
TYPE MMI_STATE:
 MMI_RESET:=0,              (* Reset = 0 *)
 MMI_INIT,                  (* Initialisation phase = 1 *)
 MMI_DISPLAY_MODE,          (* Display mode = 2 *)
 MMI_INPUT_MODE,            (* Input mode = 3 *)
 MMI_SYSTEM_MODE,           (* System mode = 4 *)
 MMI_ERROR):= MMI_RESET;    (* Error = 5 *)
END_TYPE
```

GetTextAddress
Read the text address



Function block prototype

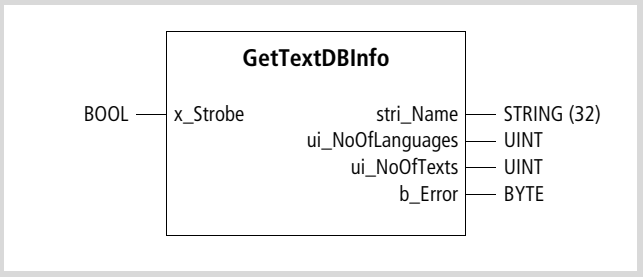
Meanings of the operands

ui_Language	Language index
ui_Text	Text index
GetTextAddress	Text address

Description

This function returns the address of the text in the text database, with the parameters that were set by the "ui_Language" and "ui_Text" inputs. If a text does not exist, the pointer produces the text "Text not available".

GetTextDBInfo
Read TextDB information



Function block prototype

Meanings of the operands

x_Strobe	The parameter is accepted with a positive edge
stri_Name	Name of the text file that was created with the XVision Tool
ui_NoOfLanguages	Number of languages used
ui_NoOfTexts	Number of texts used
b_Error	Error messages

Description

The function block reads the name of the text file and the number of languages and texts used.

A positive edge at the "x_Strobe" input produces the corresponding parameters at the outputs "stri_Name", "ui_NoOfLanguages" and "ui_NoOfTexts".

The text file is created in CoDeSys, under ‹Resources → Controller configuration → Additional parameters›, after clicking on the "Toolbox" button, and then transferred to the XV100 by using the "Load" button.

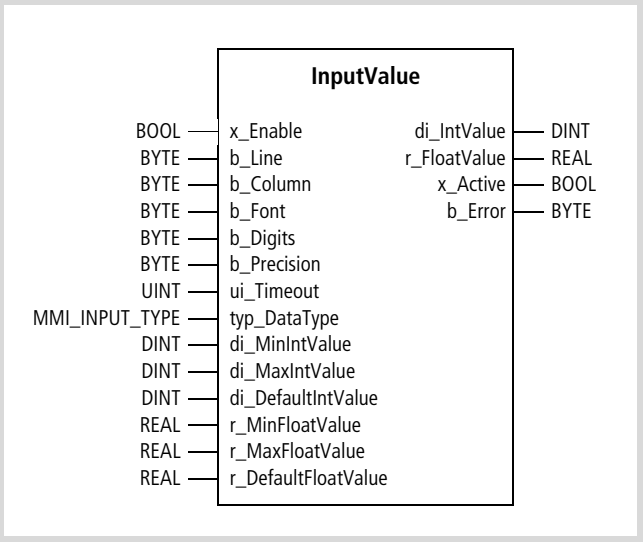
Structure of the text file:

	Language 1	**Language 2**	**. . .**	**Language n**
1	Text 1	Text 1		Text 1
2	Text 2	Text 2		Text 2
...	...	...		...
n	Text n	Text n		Text n

Error messages

0	No error
4	Text database not available

InputValue
Value entry (target value)



Function block prototype

Meanings of the operands

x_Enable	Accept the parameters	
b_Line	Line number	[0 ... 7]
b_Column	Column number	[0 ... 39]
b_Font	Number of the font set	[0 ... 4]
b_Digits	Number of digits	[1 ... 11][1)]
b_Precision	Number of decimal places (only with REAL)	[1 ... 6]
ui_Timeout	Timeout in seconds	
typ_DataType	Data type that is to be written (see data type MMI_INPUT_TYPE)	
di_MinIntValue	Minimum integer value that can be entered	
di_MaxIntValue	Maximum integer value that can be entered	
di_DefaultIntValue	Default integer value	
r_MinFloatValue	Minimum float value that can be entered	
r_MaxFloatValue	Maximum float value that can be entered	
r_DefaultFloatValue	Default float value	
di_IntValue	INTEGER input value	
r_FloatValue	REAL input value	
x_Active	Block is active	
b_Error	Error messages	

1) The number of digits depends on the data type, see table below

Data type	Max. no. of digits
UDINT	10
DINT	11 (10 digits plus sign)
REAL	8 (6 digits plus decimal point plus sign)
Password	10

Description
The function block is activated by a logic "1" at the "x_Enable" input, after which a value entry can be made (→ section "Sequence for entering values:" see below). Only one block can be active at a time.

The **"ui_Timeout"** input defined the time, in seconds, within which the entry must be concluded. If no entry is made within this time, the error message 6 (Input timeout) appears at the **"b_Error"** output. A fresh "1" signal is required at the "x_Enable" input before a new value can be entered.

The data type that is present at the "typ_DataType" input determines which value is to be entered
(see MMI_INPUT_TYPE). The parameter options are:

- unsigned double integer (UDINT)
- signed double integer (DINT)
- floating point (REAL)
- password (UDINT)

data type MMI_INPUT_TYPE[ENUM]

```
TYPE  MMI_INPUT_TYPE:
(     INPUT_UDINT:=0,
      INPUT_DINT,
      INPUT_REAL
      INPUT_PASSWORD):= INPUT_UDINT;
END_TYPE
```

The number for the font set that is to be loaded for displaying the value can be found in CoDeSys, under ‹Controller configuration → Additional parameters› in the "Visualisation" window. When using real numbers, the sign and the decimal point occupy one digit position.

Example: Digits: 8, Precision: 2

– 1 2 3 4 . 5 6

Set the parameters for the entry limits at the inputs "di_MinIntValue" and "di_MaxIntValue" or "r_MinFloatValue" and "r_MaxFloatValue".

The target values that have been entered are presented at the "di_IntValue" or "r_FloatValue" outputs.

Sequence for entering values:

- Entering an integer or real number:

A logic "1" at the "x_Enable" input activates the function block – the cursor blinks at the rightmost digit of the target value.

Use the numerical keys to enter the value. Use the ENTER key to accept the value. The "x_Active" output will be set to logic "0".

Use the "CLEAR" key to cancel the entry. A new value can be entered immediately.

Use the "ESC" key to abort the entry. Error 5 (Entry aborted) appears at the "b_Error" output. A fresh "1" signal is required at the "x_Enable" input before a new value can be entered.

For integer values:
If you enter a value that is below or above the parameter settings for the respective "di_MinIntValue" or "di_MaxIntValue" inputs, then, when you press the ENTER key, the parameterized value for the "di_DefaultIntValue" or "r_DefaultFloatValue" input will be displayed.

You can only enter a minus sign after entering at least one of the digits.

The last digit that was entered is shown at the right.

For real values:
If you enter a value that is below or above the parameter settings for the respective "r_MinFloatValue" or "r_MaxFloatValue" inputs, then, when you press the ENTER key, the parameterized value for the "r_DefaultFloatValue" input will be displayed.

You can only enter a minus sign when the decimal point has appeared.

The last digit that was entered is shown at the right.

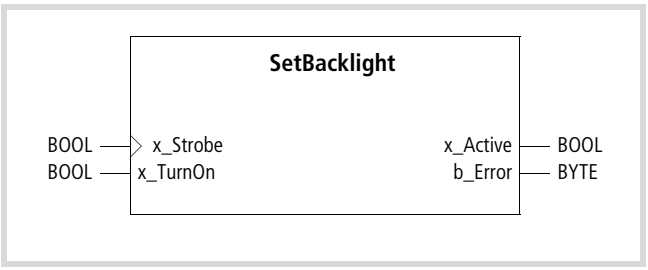
- Password entry

The procedure for entering a password is just the same as for entering an integer value. Exception: asterisks appear in the display while the entry is being made.

Example: Entry of an integer and a real number

```
PROGRAM PLC_PRG
VAR
 InputValueINT:     InputValue;
 InputValueREAL:    InputValue;
 INTVALUE:          UDINT;
 FLOATVALUE:        REAL;
 SR1:               SR;
 SR2:               SR;
END_VAR

(* Enable blocks *)
SR1(SET1:=XVFunction_F1 , RESET:=XVEnter OR XVEscape );
SR2(SET1:=XVFunction_F2 , RESET:=XVEnter OR XVEscape );

(* Parameterize blocks *)
InputValueINT(x_Enable:=SR1.Q1 , b_Line:=2 , b_Column:=6 ,
b_Digits:=6 , ui_Timeout:=10 , typ_DataType:=INPUT_DINT ,
di_MinIntValue:=-50000 , di_MaxIntValue:=50000 ,
di_DefaultIntValue:=3000 , di_IntValue=>INTVALUE );

(* Last value entered is taken as default value *)
InputValueREAL(x_Enable:=SR2.Q1 , b_Line:=4 , b_Column:=14 ,
b_Digits:=8 , b_Precision:=2 , ui_Timeout:=10 ,
typ_DataType:=INPUT_REAL , r_MinFloatValue:=-9999.99 ,
r_MaxFloatValue:=9999.99 , r_DefaultFloatValue:=FLOATVALUE ,
r_FloatValue=>FLOATVALUE );
```

SetBacklight
Switch the background lighting on/off



Function block prototype

Meanings of the operands

x_Strobe	The parameter "x_TurnOn" is accepted with a positive edge
x_TurnOn	[0,1] 0 = lighting OFF, 1 = lighting ON
x_Active	Block is active
b_Error	Error messages

Description

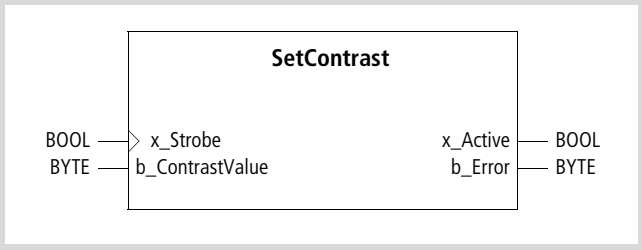
The function block switches the background lighting for the display on or off.

Depending on the state of the "x_TurnOn" input, a positive edge at the "x_Strobe" input will turn the background lighting on or off.

Error messages

0	No error
2	Over block number limit

SetContrast
Set the display contrast



Function block prototype

Meanings of the operands

x_Strobe	A positive edge accepts the parameter setting for contrast
b_ContrastValue	Contrast [0 ... 100]
x_Active	Block is active
b_Error	Error messages

Description

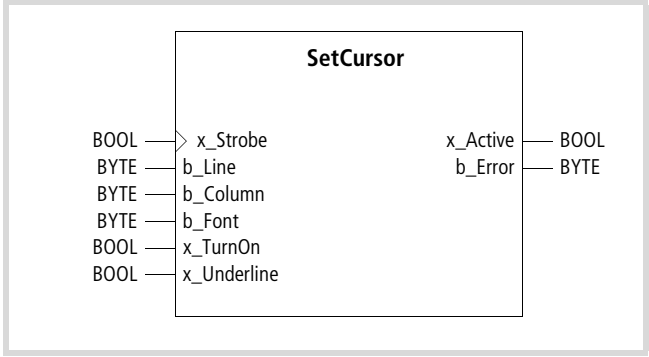
This function block sets the contrast for the display.

A positive edge at the "x_Strobe" input accepts the parameter setting for the value at the"b_ContrastValue" input. "0" indicates the lowest contrast, "100" indicates the highest contrast.

Error messages

0	No error
1	Invalid parameter
2	Over block number limit

SetCursor
Position the cursor



Function block prototype

Meanings of the operands

x_Strobe	The parameter is accepted with a positive edge	
b_Line	Line number	[0 ... 7]
b_Column	Column number	[0 ... 39]
b_Font	Number of the font set	[0 ... 4]
x_TurnOn	Cursor OFF = 0 Cursor ON = 1	
x_Underline	Cursor as block = 0 Cursor as underline = 1	
x_Active	Block is active	
b_Error	Error messages	

Description

This function block positions the cursor at the parameterized position in the display.

Depending on the state of the "x_TurnOn" input, a positive edge at the "x_Strobe" input makes the cursor visible or invisible.

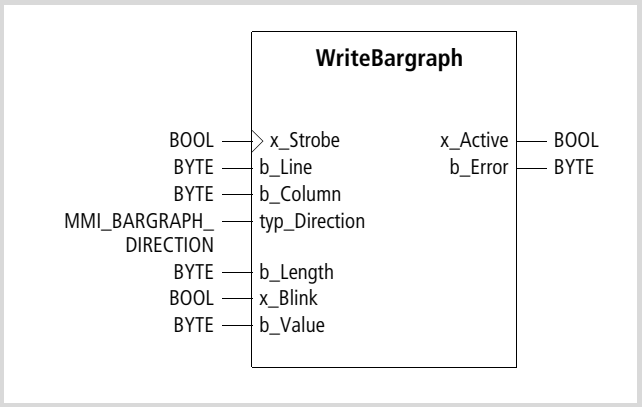
The form of the cursor can be defined at the "x_Underline" input.

The number of the font set to be loaded for displaying the cursor can be found in CoDeSys, under ‹Controller configuration → Additional parameters› in the "Visualisation" window.

Error messages

0	No error
1	Invalid parameter
2	Over block number limit

WriteBargraph
Bargraph presentation of an actual value



Function block prototype

Operand significance

x_Strobe	The parameter is accepted with a positive edge	
b_Line	Line number	[0 ... 7]
b_Column	Column number	[0 ... 39]
typ_Direction	MMI_BARGRAPH_DIRECTION	
b_Length	Length of the bargraph (in characters)	[1 ... 40][1)]
x_Blink	Blinking bargraph display	[0,1] 1 = blinking
b_Value	Bargraph value to be written	[0 ... 100] %
x_Active	Block is active	
b_Error	Error messages	

1) The bargraph is composed of small characters (small 6 × 8 pixels) or large characters (large 12 × 16 pixels), depending on the parameter at the "typ_Direction" input (see MMI_BARGRAPH_DIRECTION).

The parameter options are:

MMI_BARGRAPH_DIRECTION	Maximum length of the bargraph (in characters) with ...	
	... 4 x 20 Display	... 8 x 40 Display
SMALL_VERTICAL_BARGRAPH	Length 4	Length 8
LARGE_VERTICAL_BARGRAPH	Length 2	Length 4
SMALL_HORIZONTAL_BARGRAPH	Length 20	Length 40
LARGE_HORIZONTAL_BARGRAPH	Length 10	Length 20

Description

A positive edge at the "x_Strobe" input writes the value at the "b_Value" input to the display in the form of a bargraph.

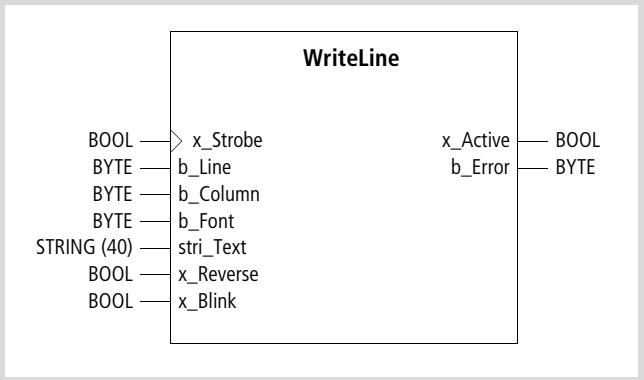
The parameter settings for the size and direction of movement for the bargraph are set at the "typ_Direction" input.

This function block displays an actual value in the form of a bargraph. The bargraph can have a vertical or horizontal direction of movement.

Error messages

0	No error
1	Invalid parameter
2	Over block number limit

WriteLine
Write a line



Function block prototype

Meanings of the operands

x_Strobe	The parameter is accepted with a positive edge	
b_Line	Line number	[0 ... 7]
b_Column	Column number	[0 ... 39]
b_Font	Number of the font set	[0 ... 4]
stri_Text	string	
x_Reverse	Inverse display of the value	[0,1] 1 = inverse
x_Blink	Blinking display of the value	[0,1] 1 = blinking
x_Active	Block is active	
b_Error	Error messages	

Description

This function block writes one line of the display.

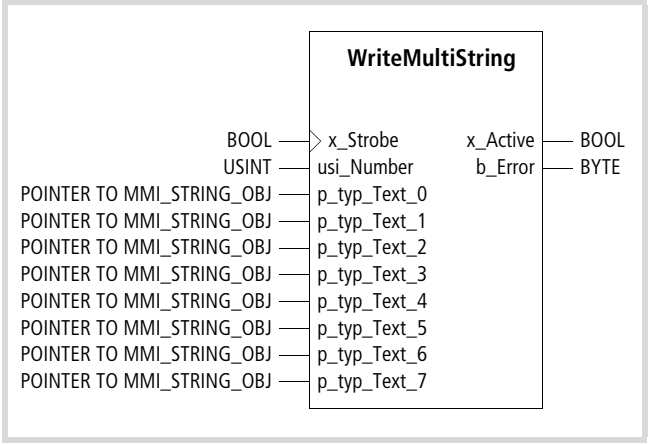
A positive edge at the "x_Strobe" input means that the alphanumeric characters that are present at the "stri_Text" input are presented at the parameterized position in the display. Any characters in the line that are in front of or behind the string will be deleted.

The number of the font set to be loaded for displaying the value can be found in CoDeSys, under ‹Controller configuration → Additional parameters› in the "Visualisation" window.

Error messages

0	No error
1	Invalid parameter
2	Over block number limit

WriteMultiString
Write up to 8 strings of alphanumeric characters



Function block prototype

Meanings of the operands

x_Strobe	The string is accepted with a positive edge
usi_Number	Number of strings [1 ... 8]
p_typ_Text_0	Pointer to the structure MMI_STRING_OBJ
p_typ_Text_1	Pointer to the structure MMI_STRING_OBJ
p_typ_Text_2	Pointer to the structure MMI_STRING_OBJ
p_typ_Text_3	Pointer to the structure MMI_STRING_OBJ
p_typ_Text_4	Pointer to the structure MMI_STRING_OBJ
p_typ_Text_5	Pointer to the structure MMI_STRING_OBJ
p_typ_Text_6	Pointer to the structure MMI_STRING_OBJ
p_typ_Text_7	Pointer to the structure MMI_STRING_OBJ
x_Active	Block is active
b_Error	Error messages

Error messages

0	No error
1	Invalid parameter
2	Over block number limit

Description
A positive edge at the "x_Strobe" input means that the function block writes up to 8 strings simultaneously to the display.

The strings are given by the structure MMI_STRING_OBJ.
The structure is composed as follows:

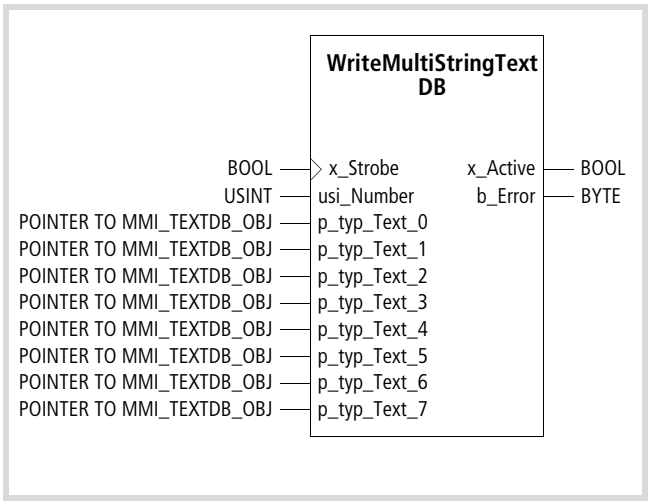
```
TYPE MMI_STRING_OBJ:
STRUCT
b_Line     :BYTE;
b_Column   :BYTE;
b_Font     :BYTE;
stri_Text  :STRING(40);
x_Reverse  :BOOL;
x_Blink    :BOOL;
b_Error    :BYTE;
END_STRUCT
END_TYPE
```

Example:

The two strings "Moeller XSystem" and "XVision Text Displays" should be presented in the first and third lines of the display, when the function key F4 is operated.

```
PROGRAM PLC_PRG
VAR
 WriteMultiString:      WriteMultiString;
 Text0:                 MMI_STRING_OBJ;
 Text1:                 MMI_STRING_OBJ;
END_VAR

(* Value transfer to the structure *)
Text0.b_Line:=0;
Text0.b_Column:=13;
Text0.b_Font:=0;
Text0.stri_Text:='Moeller XSystem';
Text0.x_Reverse:=0;
Text0.x_Blink:=0;

Text1.b_Line:=2;
Text1.b_Column:=10;
Text1.b_Font:=0;
Text1.stri_Text:='XVision Text Displays';
Text1.x_Reverse:=0;
Text1.x_Blink:=1;

(* Parameterize blocks *)
WriteMultiString(x_Strobe:=XVFunction_F4 , usi_Number:=2 ,
p_typ_Text_0:=ADR(Text0) , p_typ_Text_1:=ADR(Text1) );
```

WriteMultiStringTextDB
Write up to 8 strings simultaneously from the text file



Function block prototype

Meanings of the operands

x_Strobe	The string is accepted with a positive edge
usi_Number	Number of strings [1 ... 8]
p_typ_Text_0	Pointer to the structure MMI_TEXTDB_OBJ
p_typ_Text_1	Pointer to the structure MMI_TEXTDB_OBJ
p_typ_Text_2	Pointer to the structure MMI_TEXTDB_OBJ
p_typ_Text_3	Pointer to the structure MMI_TEXTDB_OBJ
p_typ_Text_4	Pointer to the structure MMI_TEXTDB_OBJ
p_typ_Text_5	Pointer to the structure MMI_TEXTDB_OBJ
p_typ_Text_6	Pointer to the structure MMI_TEXTDB_OBJ
p_typ_Text_7	Pointer to the structure MMI_TEXTDB_OBJ
x_Active	Block is active
b_Error	Error messages

Error messages

0	No error
1	Invalid parameter
2	Over block number limit
4	Text not available

Description
A positive edge at the "x_Strobe" input means that the function block writes up to 8 strings simultaneously to the display.
The strings that are to be written are called up from the text file through "ui_Language" and "ui_Text".

"ui_Language" and "ui_Text" for the strings that are to be shown are given by the structure MMI_TEXTDB_OBJ.

The text file is created in CoDeSys, under ‹Resources → Controller configuration → Additional parameters›, after clicking on the "Toolbox" button, and then transferred to the XV100 by using the "Load" button.

The structure is composed as follows:

```
TYPE MMI_TEXTDB_OBJ:
STRUCT
b_Line       :BYTE;
b_Column     :BYTE;
b_Font       :BYTE;
ui_Text      :UINT;
ui_Language  :UINT;
x_Reverse    :BOOL;
x_Blink      :BOOL;
b_Error      :BYTE;
END_STRUCT
END_TYPE
```

Example:

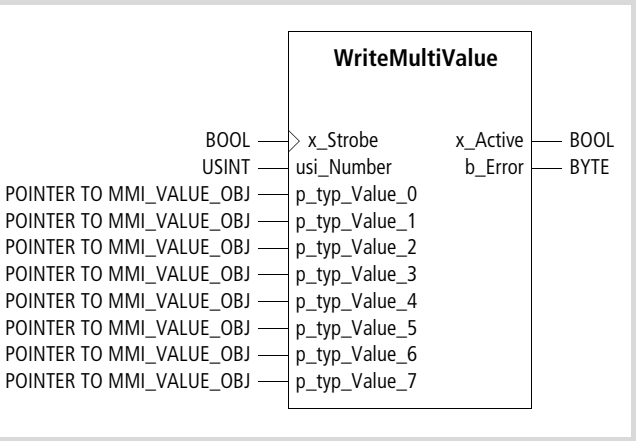
Three texts are to be displayed when the function key F1 is operated. They can then be deleted by operating the function key F2.

Layout of the text file:

	Deutsch	English	French
1	German	English	French
2			
3			
4			

```
PROGRAM PLC_PRG
VAR
 WriteMultiStringTextDB:    WriteMultiStringTextDB;
 Text0:                     MMI_TEXTDB_OBJ;
 Text1:                     MMI_TEXTDB_OBJ;
 Text2:                     MMI_TEXTDB_OBJ;
 ClearScreen:               ClearScreen;
END_VAR

(* Value transfer to the structure *)
Text0.b_Line:=0;
Text0.b_Column:=0;
Text0.ui_Language:=1;
Text0.ui_Text:=1;

Text1.b_Line:=1;
Text1.b_Column:=0;
Text1.ui_Language:=2;
Text1.ui_Text:=1;
Text1.x_Blink:=1;

Text2.b_Line:=2;
Text2.b_Column:=0;
Text2.ui_Language:=3;
Text2.ui_Text:=1;
Text2.x_Reverse:=1;

(* Parameterize blocks *)

WriteMultiStringTextDB(x_Strobe:=XVFunction_F1 ,
usi_Number:=3 ,
p_typ_Text_0:=ADR(Text0) , p_typ_Text_1:=ADR(Text1) ,
p_typ_Text_2:=ADR(Text2) );

ClearScreen(x_Strobe:=XVFunction_F2 );
```

WriteMultiValue
Write up to 8 values (actual values)



Function block prototype

Meanings of the operands

x_Strobe	The string is accepted with a positive edge
usi_Number	Number of strings
p_typ_Value_0	Pointer to the structure MMI_VALUE_OBJ
p_typ_Value_1	Pointer to the structure MMI_VALUE_OBJ
p_typ_Value_2	Pointer to the structure MMI_VALUE_OBJ
p_typ_Value_3	Pointer to the structure MMI_VALUE_OBJ
p_typ_Value_4	Pointer to the structure MMI_VALUE_OBJ
p_typ_Value_5	Pointer to the structure MMI_VALUE_OBJ
p_typ_Value_6	Pointer to the structure MMI_VALUE_OBJ
p_typ_Value_7	Pointer to the structure MMI_VALUE_OBJ
x_Active	Block is active
b_Error	Error messages

Error messages

0	No error
1	Invalid parameter
2	Over block number limit

Description
A positive edge at the "x_Strobe" input means that the function block writes up to 8 values simultaneously to the display.

The value is given by the structure MMI_VALUE_OBJ.
The structure is composed as follows:

```
TYPE MMI_VALUE_OBJ:
 STRUCT
   b_Line               :BYTE;
   b_Column             :BYTE;
   b_Font               :BYTE;
   b_Digits             :BYTE;
   b_Precision          :BYTE;
   x_LeadingZero        :BOOL;
   typ_DataType         :MMI_WRITE_TYPE;
   udi_IntValue         :UDINT
   r_FloatValue         :REAL;
   x_Reverse            :BOOL;
   x_Blink              :BOOL;
   x_Error              :BYTE;
 END_STRUCT
END_TYPE
```

The "Datatype" element has the following layout:

Data type MMI_WRITE_TYPE[ENUM]

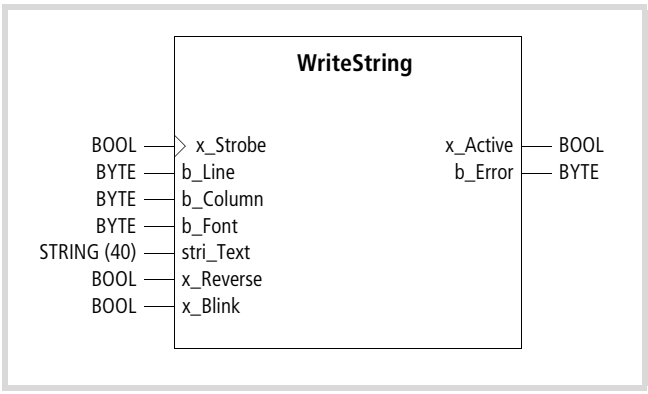
```
TYPE MMI_WRITE_TYPE:
 ( WRITE_USINT,
 WRITE_UINT,
 WRITE_SINT,
 WRITE_INT,
 WRITE_UDINT,
 WRITE_DINT,
 WRITE_REAL):=WRITE_USINT;
END_TYPE
```

Example:

Two integer values (one without and one with leading zeros) are to be shown in the display.

```
PROGRAM PLC_PRG
VAR
  WriteMultiValue:              WriteMultiValue;
  Value0:                       MMI_VALUE_OBJ;
  Value1:                       MMI_VALUE_OBJ;
  Counter:                      UINT;
END_VAR

(* Automatic count value *)
Counter:=Counter+1;
IF Counter=30000
THEN Counter:=0;
END_IF;

(* Value transfer to the structure *)
Value0.b_Line:=2;
Value0.b_Column:=19;
Value0.typ_DataType:=WRITE_UINT;
Value0.udi_IntValue:=Counter;
Value0.b_Digits:=5;

Value1.b_Line:=4;
Value1.b_Column:=17;
Value1.typ_DataType:=WRITE_UINT;
Value1.udi_IntValue:=Counter;
Value1.b_Digits:=7;
Value1.x_LeadingZero:=1;

(* Parameterize blocks *)

WriteMultiValue(x_Strobe:=TRUE , usi_Number:=2 ,
p_typ_Value_0:=ADR(Value0) , p_typ_Value_1:=ADR(Value1) );
WriteMultiValue(x_Strobe:=FALSE);
```

WriteString
Write a string



Function block prototype

Meanings of the operands

x_Strobe	The parameter is accepted with a positive edge	
b_Line	Line number	[0 ... 7]
b_Column	Column number	[0 ... 39]
b_Font	Number of the font set	[0 ... 4]
b_Text	string	
x_Reverse	Inverse display of the value	[0,1] 1 = inverse
x_Blink	Blinking display of the value	[0,1] 1 = blinking
x_Active	Block is active	
b_Error	Error messages	

Description

This function block writes a string of alphanumeric characters to the parameterized position in the display.

A positive (rising) edge at the "x_Strobe" input means that the string is shown in the display. Any characters in the line that are in front of or behind the string will not be deleted.

The number of the font set to be loaded for displaying the string can be found in CoDeSys, under ‹Controller configuration → Additional parameters› in the "Visualisation" window.

Error messages

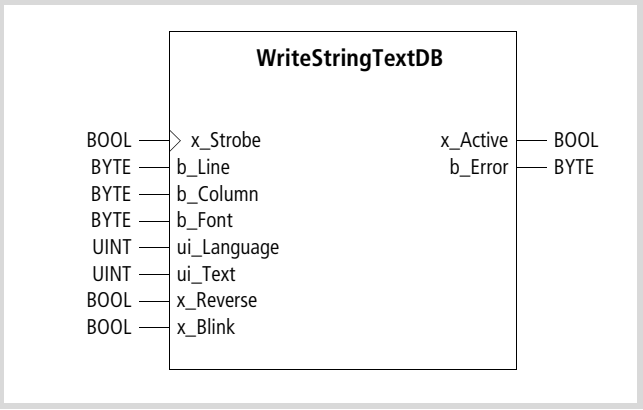
0	No error
1	Invalid parameter
2	Over block number limit

Example:

After operating the function key F2, the text "Moeller XVision" is to be shown blinking in the second line of the display.

```
PROGRAM PLC_PRG
VAR
 WriteString:                WriteString;
END_VAR

(* Parameterize blocks *)
WriteString(x_Strobe:=XVFunction_F2 , b_Line:=2 ,
b_Column:=0 , stri_Text:='Moeller XVision' , x_Blink:=1 );
```

WriteStringTextDB
Write string from text file



Function block prototype

Meanings of the operands

x_Strobe	The parameter is accepted with a positive edge	
b_Line	Line number	[0 ... 7]
b_Column	Column number	[0 ... 39]
b_Font	Number of the font set	[0 ... 4]
ui_Language	Number of the language	[1 ... n]
ui_Text	Text number	[1 ... n]
x_Reverse	Inverse display of the value	[0,1] 1 = inverse
x_Blink	Blinking text display	[0,1] 1 = blinking
x_Active	Block is active	
b_Error	Error messages	

Description

A positive edge at the "x_Strobe" input means that the function block writes a string of alphanumeric characters to the parameterized position in the display.

The string that is to be written is called up from the text file through "ui_Language" (column number) and "ui_Text" (line number).

Structure of the text file:

	Language 1	**Language 2**	**. . .**	**Language n**
1	Text 1	Text 1		Text 1
2	Text 2	Text 2		Text 2
...	...	...		...
n	Text n	Text n		Text n

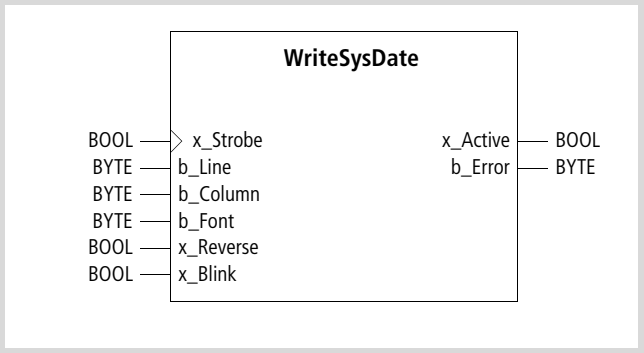
The text file is created in CoDeSys, under ‹Resources → Controller configuration → Additional parameters›, after clicking on the "Toolbox" button, and then transferred to the XV100 by using the "Load" button.

The number of the font set that is to be loaded for displaying the value can be found in CoDeSys, under ‹Controller configuration → Additional parameters› in the "Visualisation" window.

Error messages

0	No error
1	Invalid parameter
2	Over block number limit
4	Text not available

WriteSysDate
Write date



Function block prototype

Meanings of the operands

x_Strobe	The parameter is accepted with a positive edge	
b_Line	Line number	[0 ... 7]
b_Column	Column number	[0 ... 39]
b_Font	Number of the font set	[0 ... 4]
x_Reverse	Inverse display of the value	[0,1] 1 = inverse
x_Blink	Blinking display of the value	[0,1] 1 = blinking
x_Active	Block is active	
b_Error	Error messages	

Description

A positive edge at the "x_Strobe" input means that the function block writes the date to the parameterized position in the display.

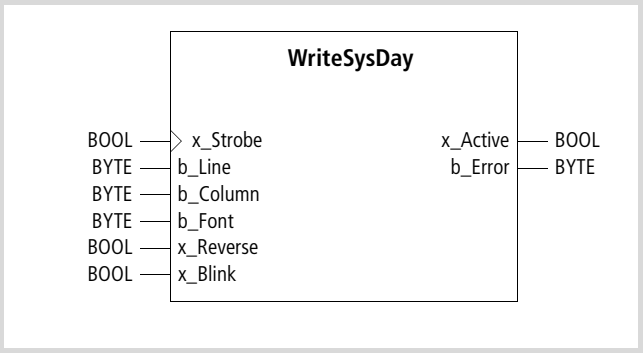
The number of the font set to be loaded for displaying the value can be found in CoDeSys, under ‹Controller configuration → Additional parameters› in the "Visualisation" window.

Format: 27.05.2002

Error messages

0	No error
1	Invalid parameter
2	Over block number limit

WriteSysDay
Write weekday



Function block prototype

Meanings of the operands

x_Strobe	The parameter is accepted with a positive edge	
b_Line	Line number	[0 ... 7]
b_Column	Column number	[0 ... 39]
b_Font	Number of the font set	[0 ... 4]
x_Reverse	Inverse display of the value	[0,1] 1 = inverse
x_Blink	Blinking display of the value	[0,1] 1 = blinking
x_Active	Block is active	
b_Error	Error messages	

Description

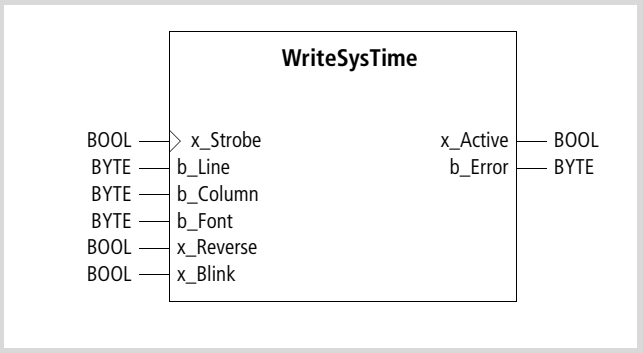
A positive edge at the "x_Strobe" input means that the function block writes the weekday to the parameterized position in the display.

The number of the font set to be loaded for displaying the value can be found in CoDeSys, under ‹Controller configuration → Additional parameters› in the "Visualisation" window.

Error messages

0	No error
1	Invalid parameter
2	Over block number limit

WriteSysTime
Write time



Function block prototype

Meanings of the operands

x_Strobe	The parameter is accepted with a positive edge	
b_Line	Line number	[0 ... 7]
b_Column	Column number	[0 ... 39]
b_Font	Number of the font set	[0 ... 4]
x_Reverse	Inverse display of the value	[0,1] 1 = inverse
x_Blink	Blinking display of the value	[0,1] 1 = blinking
x_Active	Block is active	
b_Error	Error messages	

Description

A positive edge at the "x_Strobe" input means that the function block writes the time to the parameterized position in the display.

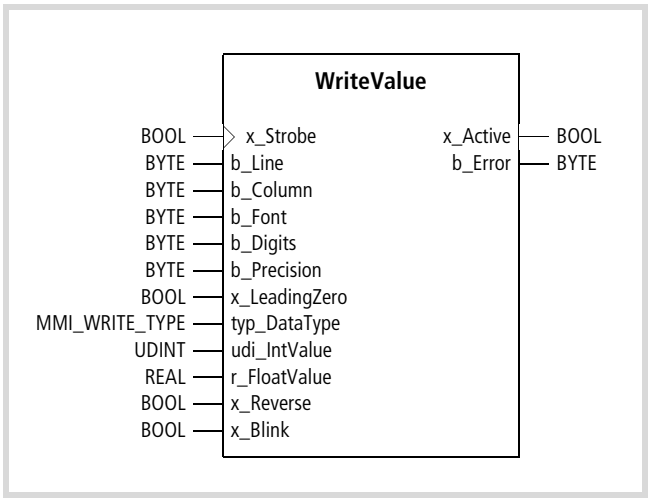
The number of the font set to be loaded for displaying the value can be found in CoDeSys, under ‹Controller configuration → Additional parameters› in the "Visualisation" window.

Format: 12:05:30

Error messages

0	No error
1	Invalid parameter
2	Over block number limit

WriteValue
Write (actual) value



Function block prototype

Meanings of the operands

x_Strobe	The parameter is accepted with a positive edge	
b_Line	Line number	[0 ... 7]
b_Column	Column number	[0 ... 39]
b_Font	Number of the font set	[0 ... 4]
b_Digits	Number of digits	[0 ... 39]
b_Precision	Number of decimal places	[0 ... 39]
x_LeadingZero	Leading zeros [0,1]	1 = with leading zeros
typ_DataType	Data type that is to be written (see data type MMI_WRITE_TYPE)	
udi_IntValue	Integer value to be written	
r_FloatValue	Real value to be written	
x_Reverse	Inverse display of the value [0,1]	1 = inverse
x_Blink	Blinking display of the value [0,1]	1 = blinking
x_Active	Block is active	
b_Error	Error messages	

Description
A positive edge at the "x_Strobe" input writes the value at the "udi_IntValue" or "ri_FloatValue" input to the display. The value at the "udi_IntValue" input is shown in "Integer" format, the value at the "r_FloatValue" input is shown in "Real" format.

The data type that is present at the "typ_DataType" input determines which value is to be displayed (see MMI_WRITE_TYPE).

The number of the font set to be loaded for displaying the value can be found in CoDeSys, under ‹Controller configuration → Additional parameters› in the "Visualisation" window.

The decimal point occupies one digit when real numbers are displayed:

Example: Digits: 6, Precision: 2

1 4 9 . 3 0

Data type MMI_WRITE_TYPE[ENUM]

```
TYPE MMI_WRITE_TYPE:
  ( WRITE_USINT,
  WRITE_UINT,
  WRITE_SINT,
  WRITE_INT,
  WRITE_UDINT,
  WRITE_DINT,
  WRITE_REAL):=WRITE_USINT;
END_TYPE
```

Error messages

0	No error
1	Invalid parameter
2	Over block number limit

Example:

A real value is to be shown with a decimal point in the third line of the display. The F1 function key is to be used to increment the value in steps of 0.1 units.

```
PROGRAM PLC_PRG
VAR
 WriteValue:           WriteValue;
 Counter:              REAL;
 Edge F1:              R_TRIG;
END_VAR

(* Edge and count value *)
EdgeF1(CLK:=XVFunction_F1);
IF EdgeF1.Q=TRUE
THEN
Counter:=Counter+0.1;
END_IF;

(* Parameterize blocks *)
WriteValue(x_Strobe:=XVFunction_F1, b_Line:=2 , b_Column:=18
, b_Digits:=4 , b_Precision:=1,
typ_DataType:=WRITE_REAL , r_FloatValue:=Counter );
```

5 Counter function blocks: counter.lib (for XIOC-1(2)CNT-100 kHz)

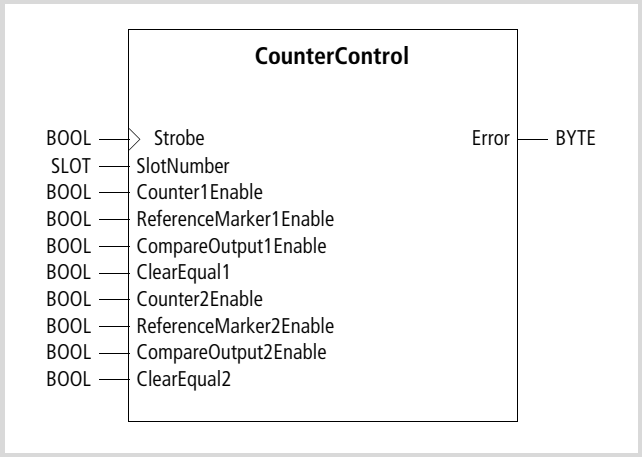
→ The library with the XC100 is only called "counter.lib"; with the XC200 it is called "XC200_counter.lib".

Five function blocks (FBs) are available for programming counter module XIOC-1(2)CNT-100kHz:

- CounterControl
- ReadCounter
- WriteCounter
- CounterFlags
- XIOC_IncEncoder

Counter function blocks

CounterControl
Enable module inputs/outputs



Function block prototype

Meanings of the operands

Strobe	The input parameters are accepted by a rising edge ("1" signal)
SlotNumber	1, 2, ... ,15 (enter full text, e. g. SLOT2)
for channel 1	
Counter1Enable	The counter function is enabled. As soon as the input is TRUE, the generator pulses will be counted. If the input is set to FALSE during operation, then the incoming pulses will not be detected.
ReferenceMarker1 Enable	In the "Linear counter" mode, the input can be enabled for the reception of the marker (reference) generator signals. Observe the polarity! The setting is made on the operating mode switch (DIP-switch); The function is not available in the "Ring counter" mode
CompareOutput1 Enable	The defined function block outputs ("=" and ">" for a linear counter; "=" for a ring counter) are enabled when TRUE is present; FALSE inhibits the outputs (they have a "0" signal).
ClearEqual1	A "1" signal at the input sets the function block outputs ("=" and ">" for a linear counter; "=" for a ring counter) and the EqualFlag (EQ) to a "0" signal. Subsequently, a "0" signal must be applied to the input, so that the output and flag can be set again.
for channel 2:	
Counter2Enable	as for channel 1
ReferenceMarker2 Enable	as for channel 1
CompareOutput2 Enable	as for channel 1
ClearEqual2	as for channel 1
Error	1: wrong slot 2: no module plugged in, or no counter 3: module does not respond

Description

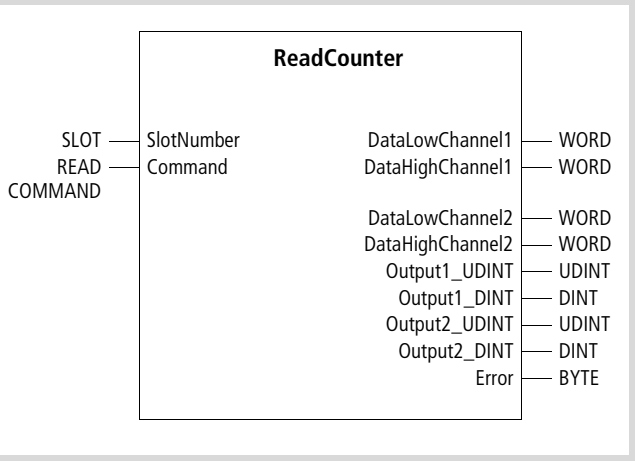
This function block is used to enable the counters (channel 1 and 2) as well as several module inputs and outputs.

Each input (see table below) has a flag assigned. If a "1" signal is applied, the functions will be performed according to the "Strobe" signal and the flags will be set. You can interrogate the flags by using the READFLAGS command of the function block CounterFlags. To make a reset, a "0" signal must be applied to the input and a positive pulse applied to the "Strobe" input.

Input designations	Flag designations
CounternEnable	CE
ReferenceMarkernEnable	ME
CompareOutputnEnable	OE
ClearEqualn	EC

All flags (apart from EC) retain their states if the state of the CPU changes from RUN → STOP or STOP → RUN.

ReadCounter
Show counter, comparison and target values



Function block prototype

Meanings of the operands

SlotNumber	1, 2, ... ,15 (enter full text, e. g. SLOT2)
Command	READCOMMAND
DataLowChannel1	lower value word, channel 1
DataHighChannel1	higher value word, channel 1
DataLowChannel2	lower value word, channel 2
DataHighChannel2	higher value word, channel 2
Output1_UDINT	value, channel1 (UDINT)
Output1_DINT	value, channel1 (DINT)
Output2_UDINT	value, channel2 (UDINT)
Output2_DINT	value, channel2 (DINT)
Error	1: wrong slot 2: no module plugged in, or no counter 3: module does not respond

Description
This function block produces, automatically and continuously – depending on the command:

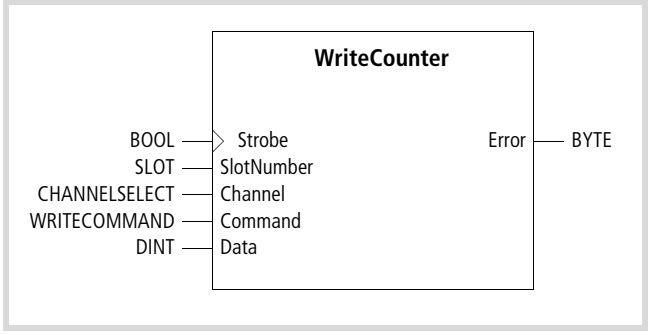
- the actual counter values (READCURRENTVALUE),
- the parameterized comparison values (READSETTINGVALUEn; n = 1, 2) or
- the parameterized target values (READPRESETVALUE).

The values are available in the form of several different data types.

The variable READCOMMAND is of type ENUM, and contains the commands (values):

- READSETTINGVALUE1: read the preset comparison value 1
- READSETTINGVALUE1: read the preset comparison value 2
- READPRESETVALUE: read the preset target value
- READCURRENTVALUE: read the actual value.

WriteCounter
Parameterizing counter, comparison and target values



Function block prototype

Meanings of the operands

Strobe	The input parameters are accepted by a rising edge ("1" signal)
SlotNumber	1, 2, ... ,15 (enter full text, e. g. SLOT2)
Channel	CHANNELSELECT
Command	WRITECOMMAND
Data	Entered value
Error	1: wrong slot 2: no module plugged in, or no counter 3: module does not respond

Description
You can use this function block – depending on the command – to set the parameters for the counter values (WRITECURRENTVALUE), the comparison value (WRITESETTINGVALUEn; n = 1, 2) or the preset target values (WRITEPRESETVALUE).

The variable CHANNELSELECT is of type ENUM, and contains the selection (values):

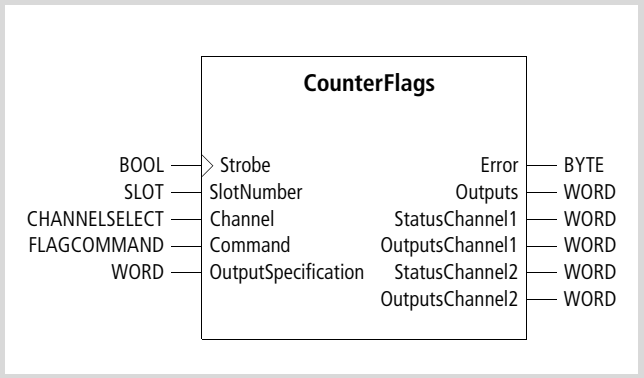
- CHANNEL1
- CHANNEL2
- BOTH

The variable WRITECOMMAND is of type ENUM, and contains the commands (values):

- WRITESETTINGVALUE1: set the comparison value 1
- WRITESETTINGVALUE1: set the comparison value 2
- WRITEPRESETVALUE: set the target value
- WRITECURRENTVALUE: set the actual value

The value is applied to the "Data" input.

CounterFlags
Activate functions and interrogate states



Function block prototype

Meanings of the operands

Strobe	The input parameters are accepted by a rising edge ("1" signal).
SlotNumber	1, 2, ... ,15 (enter full text, e. g. SLOT2)
Channel	CHANNELSELECT
Command	FLAGCOMMAND
OutputSpecification	see SPECIFYOUTPUT
Error	1 wrong slot
	2 no module plugged in, or no counter
	3 module does not respond
	4 channel active, command cannot be carried out
Outputs	see READFLAGS/Outputs
StatusChannel1	see READFLAGS/StatusChannel
OutputsChannel1	see READFLAGS/OutputsChannel
StatusChannel2	see READFLAGS/StatusChannel
OutputsChannel2	see READFLAGS/OutputsChannel

Description

You can use this function block to activate functions and interrogate states.

The variable CHANNELSELECT is of type ENUM, and contains the selection (values):

- CHANNEL1
- CHANNEL2
- BOTH

The variable FLAGCOMMAND is of type ENUM, and contains the commands (values):

- SPECIFYOUTPUT
- CLEAROVERFLOW
- CLEARUNDERFLOW
- READFLAGS

SPECIFYOUTPUT:
This command accepts the word that is applied to the "OutputSpecification" input. Precondition: the Counter-Enable input (Flag) must not be set. Every bit in this word provides information about the conditions that are involved in setting module outputs Y0, Y1, Y2 and Y3.

You can define the conditions with the aid of the following table.

Table 3: Conditions for setting the module outputs

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Comparison	>	=	>	=	>	=	>	=	>	=	>	=	>	=	>	=
Channel	2		1		2		1		2		1		2		1	
Output	Y3				Y2				Y1				Y0			

First of all, define the channel number (CH1/CH2) and the type of output. You can select a "Latch" type of output and/or a "Level" type of output.

"Latch" output (=): if the condition "Actual value = Comparison value" is met, then an output Y is set. It remains set until the "ClearEqual n" input receives a positive pulse. It is symbolised by the "=" character.

"Level" output: The output Y only produces a "1" signal while the condition "Actual value = Comparison value" is met. If the actual value falls below the comparison value, then it is reset to "0". It is symbolised by the ">" character.

Several outputs can be set by each condition ("=" or ">"). But it is not possible to define that an output, e.g. Y0, is activated when both conditions are met.

Example:

Outputs Y0 and Y1 are assigned to channel "1". Output Y0 is set if the "=" condition is met (Latch), and output Y1 is set if the ">" condition is met (Level).

The bit pattern 0021 hex must be applied to the "OutputSpecification" input:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Va-lue	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1

CLEAROVERFLOW:
The OVERFLOW flag is set if the counter goes above the maximum count value $FFFFFFFF_{hex}$ (new value: 0). The flag remains set until it is cleared by the CLEAROVERFLOW command.

You can interrogate the flag state by using the READFLAGS command of the function block CounterFlags. 16 bits are shown at the "StatusChanneln" of the function block COUNTERCONTROL.

Bit 9 (OF) indicates the state of the overflow flag.

CLEARUNDERFLOW:
The UNDERFLOW flag is set if the counter goes below the minimum count value "0" (new value: $FFFFFFFF_{hex}$).
The flag remains set until it is cleared by the CLEARUNDERFLOW command.

You can interrogate the flag state by using the READFLAGS command of the function block CounterFlags. 16 bits are shown at the "StatusChanneln" output of the "CounterControl" block.

Bit 8 (UF) indicates the state of the underflow flag.

READFLAGS:
After the command entry READFLAGS and a rising edge at the "Strobe" input, the function block outputs "Outputs", "StatusChannel n" and "OutputsChannel n" are updated. Their states are held until another transition edge occurs.

The states of "StatusChannel" and "OutputsChannel" are indicated for the channels "1" and "2" (n).

Outputs: only Bits 0 to 3 of the 16 bits have any significance:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Va-lue	0	0	0	0	0	0	0	0	0	0	0	0	Y3	Y2	Y1	Y0

Significance of the bit: Y0 to Y3:
0: output "0" signal
1: output "1" signal

StatusChannel n (n = 1, 2)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Va-lue	0	0	0	0	0	U/D	0F	UF	0	0	0	EQ	EC	OE	ME	CE

Significance of the bit:
Apart from EC, the bit states are retained if the CPU changes state, from RUN → STOP or STOP → RUN.

CE	Enable counter (default value = 0) 0: no enable 1: enable
ME	Enable marker (reference) input (default value = 0) 0: no enable 1: enable
OE	Enable output (Y) (default value = 0) 0: no enable 1: enable
EC	Clear Equal flag (default value = 0) If the function (input) "ClearEqual n" of the CounterControl block is set to TRUE, then EC n = FALSE. If it is set to FALSE, then EC = TRUE.
EQ	State of Equal flag It is set to a "1" signal if the actual value = comparison value. It remains set until a "1" signal is applied to the "ClearEqual n " input of the CounterControl block. A positive edge must be applied to the "Strobe" input to implement this action.
UF	State of Underflow flag It is set if the actual value changes from 0 to 4294967296 ($FFFFFFFF_{hex}$). It remains set until the CLEARUNDERFLOW command appears at the "Command" input of the "CounterControlFlags" block. A positive edge must be applied to the "Strobe" input to implement this command. The output words "Outputs", "StatusChannel n" and "OutputsChannel n" are set to "0".
OF	State of Overflow flag It is set if the actual value changes from 4294967296 ($FFFFFFFF_{hex}$) to 0. It remains set until the CLEARUNDERFLOW command appears at the "Command" input of the "CounterControlFlags" block. A positive edge must be applied to the "Strobe" input to implement this command. The output words "Outputs", "StatusChannel n" and "OutputsChannel n" are set to "0".
U/D	Up/Down 0: if the actual value has changed from "n" to "n – 1" 1: if the actual value has changed from "n" to "n + 1".

OutputsChannel n (n = 1, 2)

The bits contained in the word indicate the conditions on which an output depends.

Meaning of the bits

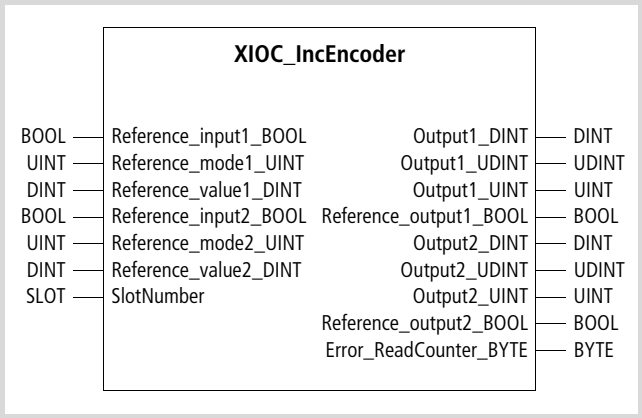
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Va-lue	0	0	>	=	0	0	>	=	0	0	>	=	0	0	>	=
Output	Y3															

Example:

0021 hex (0000 0000 0010 0001) indicates that

- output Y1 is set if the actual value > preset (target) value
- output Y0 is set if the actual value = target value.

XIOC_IncEncoder
Count pulses from incremental encoders and homing



Function block prototype

Meanings of the operands

Input	
Reference_input1_BOOL	Acceptance of the reference value for channel 1 with a rising edge (homing mode = 0) or activation (enabling) of the hardware-based homing (homing mode = 1 + 2)
Reference_mode1_UINT	Type of referencing for channel 1:
	0: by rising edge on Reference_input1_BOOL
	1: once by hardware after activation of Reference_input1_BOOL
	2: permanent by hardware after activation of Reference_input1_BOOL
Reference_value1_DINT	Start value (32 Bit) of the counter for channel 1
Reference_input2_BOOL	Acceptance of the reference value for channel 2 with a rising edge (homing mode = 0) or activation (enabling) of the hardware-based homing (homing mode = 1 + 2)
Reference_mode2_UINT	Type of referencing for channel 2:
	0: by rising edge on Reference_input2_BOOL
	1: once by hardware after activation of Reference_input2_BOOL
	2: permanent by hardware after activation of Reference_input2_BOOL
Reference_value2_DINT	Start value (32 Bit) of the counter for channel 2
SlotNumber :	Slot number of the XI/OC counter module: 1, 2, ... ,15 (enter full text, e. g. SLOT2)

Output	
Output1_DINT	Current 32 Bit counter state, bipolar for channel 1
Output1_UDINT	Current 32 Bit counter state, unipolar for channel 1
Output1_UINT	Current 16 Bit counter state, unipolar for channel 1
Reference_output1_BOOL	Hardware referencing activated for channel 1:
	1: Hardware referencing activated (reference mode 1 and 2)
	0: Hardware referencing deactivated, as it has been initiated once (referencing mode 1)
Output2_DINT	Current 32 Bit counter state, bipolar for channel 2
Output2_UDINT	Current 32 Bit counter state, unipolar for channel 2
Output2_UINT	Current 16 Bit counter state, unipolar for channel 2
Reference_output2_BOOL	Hardware referencing activated for channel 2:
	1: Hardware referencing activated (reference mode 1 and 2)
	0: Hardware referencing deactivated, as it has been initiated once (referencing mode 1)
Error_ReadCounter_BYTE	1: wrong slot
	2: no module plugged in, or no counter
	3: module does not respond

→ The hardware with the setting possibilities for the mode and the potential assignment of the inputs is described in the manual "XIOC Signal Modules" (MN05002002Z-EN; previously AWB2725-1452GB).
This manual is available online, as a PDF file, at:
http://www.eaton.com/moeller → **Support**
Enter the manual number here as the search text.

Description

This function block counts the pulses from the incremental encoders which are connected to the XIOC-1CNT-100KHZ or XIOC-2CNT-100KHZ modules. The following functions can be run:

- Initiate referencing
- Define referencing type
- Preset of the counter value at the point of zero crossover (home position)
- Read the count values of the incremental encoders.

Referencing for 2 channels (2 incremental encoders) can be implemented with the function block. The functions for channel 1 are described in the following. The same functions are available for channel 2 (only available with XIOC-2CNT-100KHZ).

The following referencing modes can be selected with the "Reference_mode1_UINT" input:

0 = Referencing with software with each rising edge of "Reference_input1_BOOL"

1 = Once-off referencing by hardware after setting "Reference_input1_BOOL = 1"

2 = Permanent referencing by hardware (with each incremental encoder marker pulse) after setting of "Reference_input1_BOOL = 1"

→ Referencing by hardware occurs when the incremental encoder issues a pulse (usually once per rotation). The signal must be applied to the XIOC-CNT1/2-100KHZ module.

Referencing mode 1:
If the referencing procedure has been activated, the count value is set initially to 1000000000 for technical reasons. The "Reference_output1_BOOL" output indicates with the "1" state that hardware-based referencing has been activated.
During referencing (when the incremental encoder issues the marker pulse signal) the count value is initialized with the "Reference_value1_DINT" (referencing mode 0 + 1 + 2).
The Reference_output1_BOOL" output has the "0" state.

Referencing mode 2:
The "Reference_output1_BOOL" output remains permanently set to "1", as the hardware referencing remains permanently active.

The count value is output in varying data formats from the "Output1_DINT", "Output1_UDINT" and "Output1_UINT" outputs.

→ The Motion-Control-Toolbox is available for the motion control topic with the manual MN05010005Z-EN (previously AWB2700-1454GB).

Application example:

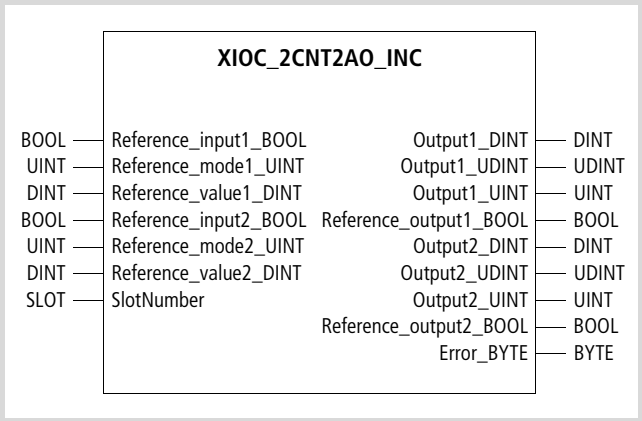
```
PROGRAM Test_XIOC_Counter_module
VAR
   XIOC_Counter_module_slot1 : XIOC_IncEncoder ;
   Falling_edge : F_TRIG ;
   Activate_referencing_with_hardware : BOOL ;
   Actual_position : DINT ;
   Hardware_referencing_activated : BOOL ;
   Referencing_finished : BOOL ;
END_VAR

   CAL  XIOC_Counter_module_slot1(
        Reference_input1_BOOL:=Activate_referencing_with_
        hardware ,
        Reference_mode1_UINT:=1 ,
        Reference_value1_DINT:=10000 ,
        SlotNumber:=Slot1 ,
        Output1_DINT=>Actual_position )

   LD   XIOC_Counter_module_slot1.Reference_output1_BOOL
   ST   Hardware_referencing_activated

   LD   XIOC_Counter_module_slot1.Reference_output1_BOOL
   ST   Falling_edge.CLK
   CAL  Falling_edge
   LD   Falling_edge.Q
   S    Referencing_finished

END_PROGRAM
```

6 Counter elements: Counter_Analog.lib (for XIOC-2CNT-2AO-NC)

Two function blocks (FBs) are available for programming the analog counter module XIOC-2CNT-2AO-INC:

- Incremental encoder evaluation
- Analog value output.

XIOC_2CNT2AO_INC incremental encoder evaluation



Function block prototype

Meanings of the operands

Input	
Reference_input1_BOOL	Acceptance of the reference value for channel 1 with a rising edge (homing mode = 0) or activation (enabling) of the hardware-based homing (homing mode = 1 + 2)
Reference_mode1_UINT (→ section "Settings" on page 60)	Type of referencing for channel 1:
	0: by rising edge on Reference_input1_BOOL
	1: once by hardware after activation of Reference_input1_BOOL
	2: permanent by hardware after activation of Reference_input1_BOOL
Reference_value1_DINT	Start value (32 bit) of the counter for channel 1 after referencing
Reference_input2_BOOL	Acceptance of the reference value for channel 2 with a rising edge (homing mode = 0) or activation (enabling) of the hardware-based homing (homing mode = 1 + 2)
Reference_mode2_UINT (→ section "Settings" on page 60)	Type of referencing for channel 2:
	0: by rising edge on Reference_input2_BOOL
	1: once by hardware after activation of Reference_input2_BOOL
	2: permanent by hardware after activation of Reference_input2_BOOL
Reference_value2_DINT	Start value (32 bit) of the counter for channel 2 after referencing
SlotNumber :	Slot number of the XI/OC counter module : 1, 2, ... ,15 (enter full text, e. g. SLOT2)
Output	
Output1_DINT	Current 32 Bit counter state, bipolar for channel 1
Output1_UDINT	Current 32 Bit counter state, unipolar for channel 1
Output1_UINT	Current 16 Bit counter state, unipolar for channel 1
Reference_output1_BOOL	Hardware referencing activated for channel 1:
	1: Hardware referencing activated (reference mode 1 and 2)
	0: Hardware referencing deactivated, as it has been initiated once (referencing mode 1)
Output2_DINT	Current 32 Bit counter state, bipolar for channel 2
Output2_UDINT	Current 32 Bit counter state, unipolar for channel 2
Output2_UINT	Current 16 Bit counter state, unipolar for channel 2
Reference_output2_BOOL	Hardware referencing activated for channel 2:
	1: Hardware referencing activated (reference mode 1 and 2)
	0: Hardware referencing deactivated, as it has been initiated once (referencing mode 1)
Error_BYTE	1: wrong slot
	2: no module plugged in, or no counter

→ The hardware with the setting possibilities for the mode and the potential assignment of the inputs is described in the manual "XIOC Signal Modules" (MN05002002Z-EN; previously AWB2725-1452GB). This manual is available online, as a PDF file, at http://www.eaton.com/moeller → **Support**.
Enter the manual number here as the search text.

→ If you use this function block, the inputs of the XIOC_2CNT2AO_INC may not be declared or accessed.

Settings
The type of referencing is set on the "Reference_inputx_BOOL" input. To ensure that this setting becomes valid, enter the following values in the configurator of the CoDeSys in the "Other parameters" register:

– Number of references: Once off
– Edge evaluation: factor 1, 2 or 4 (factor 4 = highest resolution)
– Reference value: 0

Description
The function block counts the pulses from the incremental encoders which are connected to the XIOC-2CNT-2AO-INC module. The following functions can be run:

- Initiate referencing
- Define referencing type
- Preset of the counter value at the point of zero crossover (home position)
- Read the count values of the incremental encoders.

Referencing for 2 channels (2 incremental encoders) can be implemented with the function block. The functions for channel 1 are described in the following. The same functions are available for channel 2.

The following referencing modes can be selected with the "Reference_mode1_UINT" input:

0 = Referencing with software with each rising edge of "Reference_input1_BOOL"

1 = Once-off referencing by hardware after setting "Reference_input1_BOOL = 1"

2 = Permanent referencing by hardware (with each incremental encoder marker pulse) after setting of "Reference_input1_BOOL = 1"

→ Referencing by hardware occurs when the incremental encoder issues a pulse (usually once per rotation). The signal must be applied to the XIOC-2CNT-2AO-INC module.

Referencing mode 1:
The "Reference_output1_BOOL" output indicates with the "1" state that hardware-based referencing has been activated. During referencing (when the incremental encoder issues the marker pulse signal) the count value is initialized with the "Reference_value1_DINT" (referencing mode 0 + 1 + 2). The "Reference_output1_BOOL" output has the "0" state.

Referencing mode 2:
The "Reference_output1_BOOL" output remains permanently set to "1", as the hardware referencing remains permanently active.

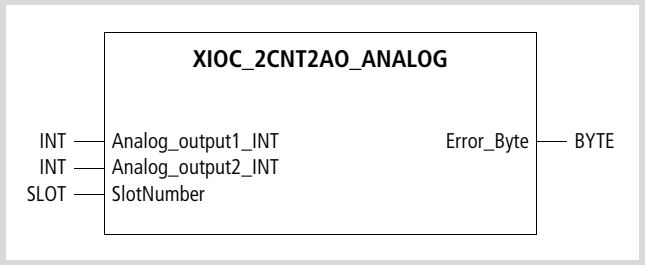
The count value is output in varying data formats from the "Output1_DINT", "Output1_UDINT" and "Output1_UINT" outputs.

→ The Motion-Control-Toolbox is available for the motion control topic with the manual MN05010005Z-EN (previously AWB2700-1454GB).

Application example:

```
Program: Test_XIOC_2CNT2AO_INC

VAR
    Activate_hardware_referencing1_BOOL : BOOL ;
    Activate_hardware_referencing2_BOOL : BOOL ;
    Actual_position1_DINT : DINT ;
    Actual_position2_DINT : DINT ;
    Speed_drive1_INT : INT ;
    Speed_drive2_INT : INT ;
    Error_BYTE : BYTE ;
    Error_analog_BYTE : BYTE;
    XIOC_2cnt2ao_inc_01_and_02 : XIOC_2cnt2ao_inc;
    XIOC_2cnt2ao_analog_01_and_02 : XIOC_2cnt2ao_analog ;
END_VAR

(*zone1*)

CAL XIOC_2cnt2ao_inc_01_and_02(
    Reference_input1_BOOL:=Activate_hardware_referencing1_
    BOOL ,
    Reference_mode1_UINT:=1 ,
    Reference_value1_DINT:=0 ,
    SlotNumber:=Slot1 ,
    Output1_DINT=>Actual_position1_DINT ,
    Error_BYTE=>Error_BYTE )

(* Cal function_block_for_positioning_01 *)
CAL XIOC_2cnt2ao_analog_01_and_02(
    Analog_output1_INT:=Speed_drive1_INT ,
    SlotNumber:=Slot1 ,
    Error_BYTE=>Error_Analog_BYTE)

(*zone2*)

CAL XIOC_2cnt2ao_inc_01_and_02(
    Reference_input2_BOOL:=Activate_hardware_referencing2_
    BOOL ,
    Reference_mode2_UINT:=1 ,
    Reference_value2_DINT:=0,
    SlotNumber:=Slot1 ,
    Output2_DINT=>Actual_position2_DINT )

(* Cal function_block_for_positioning_02 *)
CAL XIOC_2cnt2ao_analog_01_and_02(
    Analog_output2_INT:=Speed_drive2_INT ,
    SlotNumber:=Slot1)
```

XIOC_2CNT2AO_ANALOG analog value output



Function block prototype

Meanings of the operands

Analog_output1(2)_INT	Decimal integer value
SlotNumber	Slot number of the XI/OC counter analog module: 1, 2, ... ,15 (enter full text, e.g. SLOT2)
Error_Byte	1: incorrect slot
	2: no module plugged in, or no counter

Description

The decimal value set on the Analog_output 1(2)_INT inputs is output via the analog outputs of the XIOC-2CNT-2AO-INC module.

Decimal value (function block input)	Analog value (function block output)
0	0 V
2047	10 V
– 1	– 0.005 V
– 2048	– 10 V

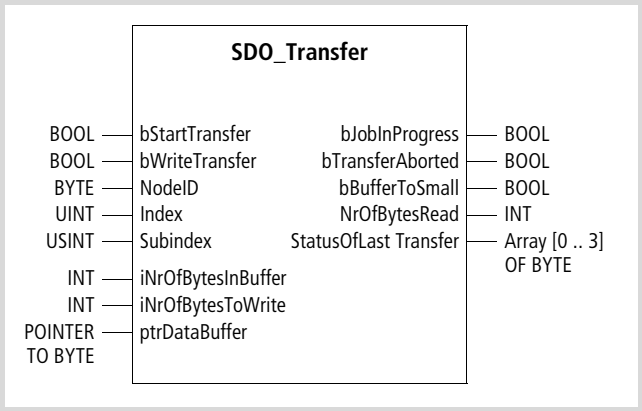
Application example

```
Program: Test_XIOC_2CNT2AO_ANALOG

VAR
    output1_5V : INT :=1023
    output2_minus_10V : INT := - 2048;
    Error_BYTE : BYTE;
    XIOC_2cnt2ao_analog_01_and_02 : XIOC_2cnt2ao_analog ;
END_VAR

CAL XIOC_2cnt2ao_analog_01_and_02(
    Analog_output1_INT:=output1_5V ,
    Analog_output2_INT:=output2_minus_10V ,
    SlotNumber:=Slot1 ,
    Error_BYTE=>Error_BYTE )
```

7 Transfer block: CANopen_Utilities.lib

Transfer block

SDO_Transfer (XC-100)
Transfer parameters



Function block prototype

Meanings of the operands

bStartTransfer	A positive edge starts an SDO transfer. Prerequisites: • the block output "bJobInProgress" must have the state "FALSE", • the corresponding network node is in the "Operational" state.	
bWriteTransfer	True:	Download (transfer) data
	False:	Upload (receive) data
NodeID	Node-ID for the network node to be addressed	
Index	Address index	
SubIndex	Address subindex	
INrOfBytesInBuffer	Number of bytes available in the data buffer (only for upload)	
INrOfBytesToWrite	Number of data bytes to be written (only for download)	
PtrDataBuffer	Pointer to a data buffer that will receive the bytes being sent or received.	
bJobInProgress	False:	No transfer in progress, the result of the previous transfer can be read out.
	True:	SDO transfer is in progress. A new transfer cannot be started. No access to the data buffer!
bTransferAborted	True:	If the last SDO transfer was aborted with an error message. The abort code (as per CIA-DSP 301) can be read from "StatusOfLastTransfer".
bBufferToSmall	True:	If the originating data buffer is too small to hold the data bytes.
NrOfBytesRead	Number of data bytes read out (only for upload)	
StatusOfLast Transfer	If the bTransferAborted flag is set, the abort code (as per CIA-DSP 301) can be read out here.	

→ Only one instance of the SDO function block can be created for each network node to be addressed.

A maximum of 248 bytes can be sent or received.

Description

You can use the SDO block (SDO = ServiceDataObject) to access the object directory of a network node.
SDOs are mainly used to configure the network nodes.

Abort code

Abort code	Description
0503 0000h	Toggle bit does not change
0504 0000h	Timeout for SDO protocol
0504 0001h	Unknown Client/Server command
0504 0002h	Invalid block length (block mode)
0504 0003h	Invalid sequence number (block mode)
0504 0004h	CRC error (block mode)
0504 0005h	Invalid memory range
0601 0000h	Unsupported object access
0601 0001h	Cannot read out a "Write" object
0601 0002h	Cannot write to a "Read" object
0602 0000h	Object is not in object directory
0604 0041h	Object cannot be attached to the PDO
0604 0042h	The number and length of the object to be mapped exceed the length of the PDO.
0604 0043h	General parameter incompatibility
0604 0047h	Internal device incompatibility
0606 0000h	Access causes a hardware error
0607 0010h	Wrong data type, wrong length of Service parameter
0607 0012h	Wrong data type, Service parameter too long
0607 0013h	Wrong data type, Service parameter too short
0609 0011h	Subindex missing
0609 0030h	Outside parameter range (write access)
0609 0031h	Above parameter range
0609 0032h	Below parameter range
0609 0036h	Maximum value is smaller than minimum value
0800 0000h	General error
0800 0020h	Data cannot be transferred to the application for the target device or stored there.
0800 0021h	Data cannot be transferred to the application or stored. Cause: error in the application for the target device
0800 0022h	Data cannot be transferred to the application or stored. Cause: prevented by current status of target device.
0800 0023h	Dynamic generation of the object directory is faulty, or the object directory is missing.

8 Data access function blocks for files: XC100_File.lib

The library contains the following function blocks:

- File Open
- File Close
- File Read
- File Write
- File Delete
- File Rename
- File Set Pos
- File Get Size

→ These function blocks can be used only for the XC100.

Description of the blocks

These function blocks enable access of the multimedia memory card (MMC) file system. Up to four files can be opened simultaneously.

Subdirectories are supported. However, the path may not exceed 80 characters in length.

In order to avoid inconsistencies in the file system, the operating system closes files which have been opened by the user program when a PLC status change to "STOP" occurs.

→ If the Multimedia memory card is pulled out when a function block is active, it is possible that this card can no longer be accessed.
The card will have to be reformatted to correct the fault.

Fundamental block handling procedures

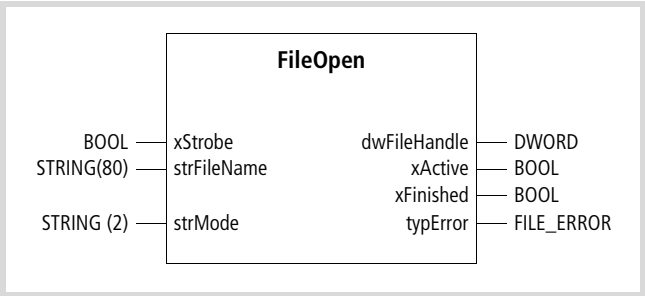
When a positive edge is detected on the "Strobe" input of the function block, the parameters of the inputs are accepted and the process is started.

The "Active" output defines the state of the function block:

High	The process is run.
High -> Low -edge	The processing is complete and the output data is valid, if a "0" is present on the error output.

The error output issues a code with faulty processing. This provides information concerning the nature of the fault
(see table 4 on page 69).

FileOpen



Function block prototype

Meanings of the operands

XStrobe	Start
strFileName	Name of the file
strMode	Mode in which the file is to be opened
dwFileHandle	File handle which is required for access to further access function blocks
xActive	Active output
xFinished	When the function block has finished, the output carries a "1" signal (negated xActive output).
typError	Error output (see table 4 on page 69)

Description

This function block opens a file on the memory card.

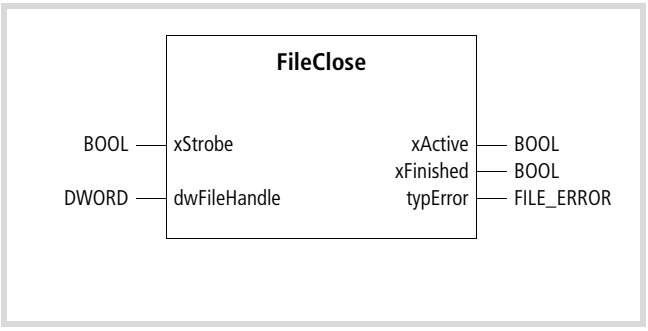
→ A file can only be opened once.

You determine the mode in which the file is to be opened with the "w, r, rw, or a" code on the "strMode" input.

w = Write	The file is opened in write mode. An existing file with this name will be overwritten.
r = Read	"r" mode opens a file for sequential reading. With every read access, the read position is pushed forward by the number of bytes read.
rw = Read and write	The file is overwritten or created. The contents of the file will be erased during opening.
a = Append	A file is opened for writing. In order to write the data at the end of a file, the file pointer must first of all be set to the end of the file.

▽ **Important**
Opening a file assigned with "w" and reclosing it is sufficient to overwrite the file and to create a file with 0 bytes in length.

FileClose



Function block prototype

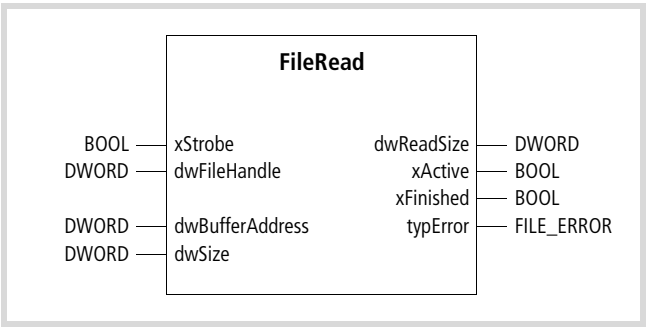
Meanings of the operands

XStrobe	Start
dwFileHandle	File handle of the "FileOpen" function block
xActive	Active output
xFinished	When the function block has finished, the output carries a "1" signal (negated xActive output)
typError	Error output (see table 4 on page 69)

Description

This function block closes an open file.
The file content is consistent after closing.

FileRead



Function block prototype

Meanings of the operands

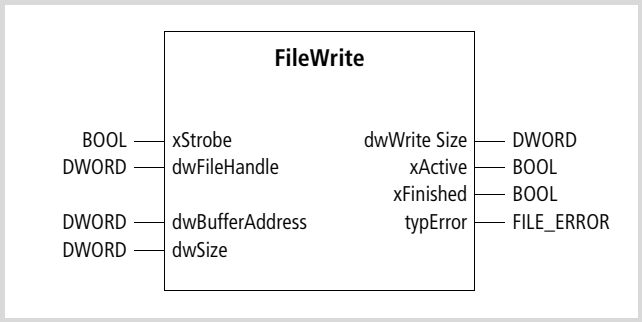
XStrobe	Start
dwFileHandle	File handle of the "FileOpen" function block
dwBufferAddress	Address (ADR) to a buffer
dwSize	Number of bytes to be read
dwReadSize	Number of bytes read
xActive	Active output
xFinished	When the function block has finished, the output carries a "1" signal (negated xActive output)
typError	Error output (see table 4 on page 69)

Description

This function block reads data from the opened file.

The file pointer is at the beginning of the file after opening and moves forward by the number of bytes read with each read access. It is not possible to read beyond the end of the file. If the file pointer is positioned at the end of the file, the function block indicates "0" bytes for the number of bytes read.

FileWrite



Function block prototype

Meanings of the operands

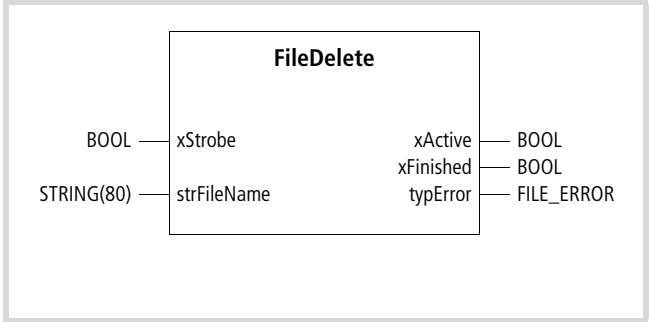
XStrobe	Start
dwFileHandle	File handle of the "FileOpen" function block
dwBufferAddress	Address (ADR) to a buffer
dwSize	Number of bytes to be written
dwWriteSize	Number of bytes written
xActive	Active output
xFinished	When the function block has finished, the output carries a "1" signal (negated xActive output)
typError	Error output (see table 4 on page 69)

Description

This function block writes data into an open file.

The file pointer is at the beginning of the file after opening and moves forward by the number of bytes written with each write access.

FileDelete



Function block prototype

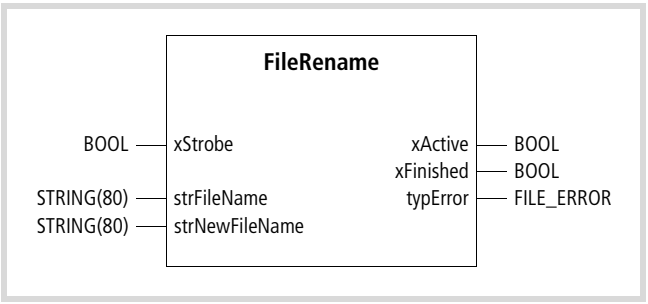
Meanings of the operands

XStrobe	Start
strFileName	Name of the file
xActive	Active output
xFinished	When the function block has finished, the output carries a "1" signal (negated xActive output)
typError	Error output (see table 4 on page 69)

Description

This function block deletes a file from the memory card.
It is only possible to delete a closed file.

FileRename



Function block prototype

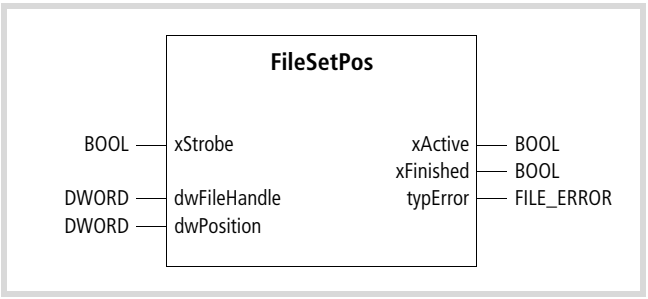
Meanings of the operands

XStrobe	Start
strFileName	existing name of the file
strNewFileName	new name of the file
xActive	Active output
xFinished	When the function block has finished, the output carries a "1" signal (negated xActive output)
typError	Error output (see table 4: Error codes)

Description

This function block renames the file.

FileSetPos



Function block prototype

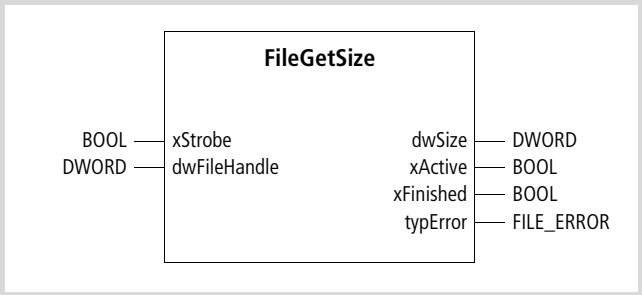
Meanings of the operands

XStrobe	Start
dwFileHandle	File handle of the "FileOpen" function block
dwPosition	Offset in bytes from the start of the file
xActive	Active output
xFinished	When the function block has finished, the output carries a "1" signal (negated xActive output)
typError	Error output (see table 4: Error codes)

Description

This function block sets the file pointer at any required position within an open file. It is not possible to set it beyond the end of the file.

FileGetSize



Function block prototype

Meanings of the operands

XStrobe	Start
dwFileHandle	File handle of the "FileOpen" function block
dwSize	Size of the file in bytes
xActive	Active output
xFinished	When the function block has finished, the output carries a "1" signal (negated xActive output)
typError	Error output (see table 4: Error codes)

Description

The current size of an opened file is defined at the "udiSize" function block output.

Error codes

Table 4: Error codes

Function block type: File_	Code	Description
All	0x00	NoError No error
Open, Delete, GetSize, Rename	0x01	File not found File not found
Open	0x02	Too many open files (> 4) To many files open (> 4)
Open	0x03	File already opened File already open
Open, Delete	0x04	Access denied Access denied
Open, Close, Write, Read, Delete, Rename	0x05	MMC access denied MMC Access denied
Close, Write, Read, SetPos	0x06	Invalid file handle Invalid file handle
SetPos	0x07	Invalid position Invalid position
Rename	0x08	File already exists File already exists
Open	0x0a	Invalid file mode Invalid file mode
All	0x0f	More than 10 function blocks are already activated Queue full

9 Acyclic file access blocks for PROFIBUS-DP: xSysNetDPMV1.lib

The library contains the following function blocks:

- XDPMV1_READ
- XDPMV1_WRITE

→ The blocks are for use with the XC100 and XC200.

Description of the blocks

Objects (data packages) from a slave can be read acyclically with the XDPMV1_READ block and objects can be written acyclically to a slave with the XDPMV1_WRITE block. A prerequisite is that the slave must support the PROFIBUS-DP DP-V1 protocol type and cyclic data transfer must occur.

The data of the slaves which can be read or written acyclically are described in the following as DP-V1 objects. The attributes of the DP-V1 object to be processed in the slave should be taken from the respective slave documentation. The address of a DP-V1 object is defined by the slot number (uiSlot), the index (uiIndex) and the read/write data length.

Please observe the following guidelines when using these blocks:

- Do not access these blocks cyclically as it will unnecessarily load cyclic bus operation.
- Program a maximum of one block to read and one block to write for each master.
- Interlock the read and write blocks so that only one block can be active at a time!

The slaves must be accessed consecutively via the blocks. DP-V1 objects of a slave must be processed consecutively.

Fundamental block handling procedures

The "Read" output must have a "1 signal" state in order to start a job.

▶ Apply a "1 signal" (positive/rising edge) to the "Enable" input.

The function block is activated and the "Ready" output indicates the "0" state. After the job is complete the "Ready" output will be set to the "1" state.

▶ After the "0" → "1" signal transition, scan output "typState" → "Done".
▶ Access the V1State outputs in order to analyze the actual block state:

If for example "Done" = TRUE the job has been correctly completed. A fault has occurred if a "False" signal is present.

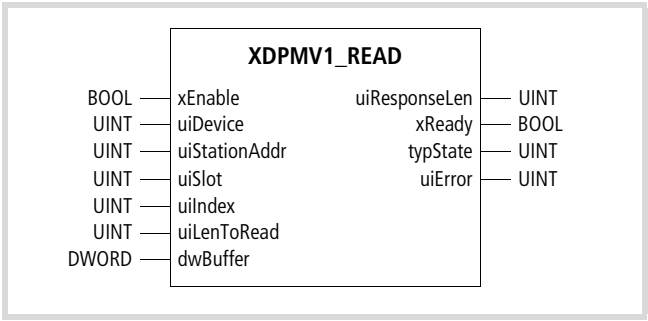
▶ Access the "typState" output → "DoneWithError" and "InvalidParam" in order to specify the fault:

DoneWithError =TRUE:	Read the Error output and a fault code is displayed (→ table 6).
InvalidParam =TRUE:	Block inputs are incorrectly parameterized

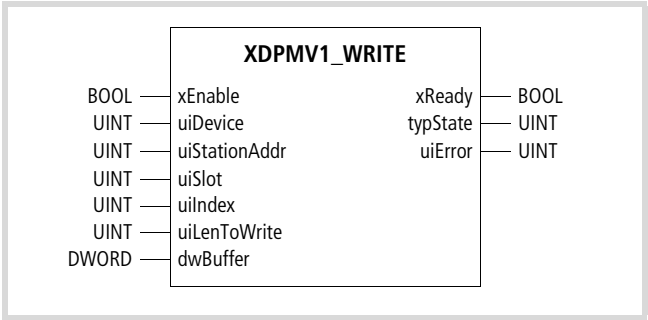
The "typState" output indicates the block state. The individual states are defined in the "typState" variables. They can assume the following values:

```
0: not being processed
1: invalid parameter
2: started
3: job successfully completed
4: Job ended with an error→ evaluate the error output
```

XDPMV1_READ, XDPMV1_WRITE



Function block prototype



Function block prototype

Meanings of the operands

xEnable	Start
uiDevice	Device number → table 5
ui StationAddr	Station address
uiSlot	Slot number in the slave (value range 0 to 254)
uiIndex	Index in the slave (value range 0 to 254)
uiLenToRead	Length of read data in bytes (value range 0 to 240). The length of the read data is dependent on the slave (see slave documentation). If the read data length is unknown, enter the maximum value of "240". The actual read byte length is indicated by the "uiResponseLen" output.
uiLenToWrite	Length of write data in bytes (value range 0 to 240). The length of the write data is dependent on the slave (see slave documentation). An error message appears if you do not enter the exact value.
dwBuffer	Address to a buffer
uiResponseLen	Length of data actually read (bytes)
xReady	Job process state
typState	State of function block (→ page 71)
uiError	Error code (→ table 6)

Function block assignment (Device number)

Up to three DP function blocks can be used with the XC200. Each of these DP function blocks can use a block for acyclic read and a block for acyclic write. All in all a maximum of six function blocks can be used in the user program. Using device numbers you determine which function block is assigned to which DP module. The device number depends on the DP module slot and is defined in table 5.

With the XC100 the device number is generally "0" as only one DP module can be used by this control.

Table 5: Device number for XC200

XI/OC slot	**1**	**2**	**3**	
Module	DP-M	DP-M	DP-M	
Device No.	0	1	2	
Module	DP-M	DP-M	X-module	
Device No.	0	1	–	
Module	X-module	DP-M	DP-M	
Device No.	–	0	1	
Module	DP-M	X-module	DP-M	Configuration fault: Voids are invalid!
Device No.	0	–	2	
Module	X-module	X-module	DP-M	
Device No.	–	–	0	

X-module: Module (No PROFIBUS-DP module)

Error code on "Error" output

Table 6: Error codes

2	No resources for order processing are available in the slave (internal slave fault)
3	The master has not activated the DP-V1 mode for this slave. Check the DP configuration.
9	Invalid response (internal slave fault)
17	No response from this slave. Possible causes: • Incorrect StationAddress • Slave or bus are not active
18	General bus fault • Check bus cables • Check master • Check the DP address or high station address from further masters in the configuration
25	Unintelligible answer, the slave is not conform to the DP-V1 standard
54	Incorrect answer
129	DP-V1 communication has not been configured and activated or the slave address does not exist
130	DP-V1 communication has been inhibited, the answer of the previously addressed slave is false
131	A job is still active (internal FB error)
132	Parameter and data fault (internal fault)
133	Parameter fault. Possible causes: StationAddress, SlotNumber or Index are false

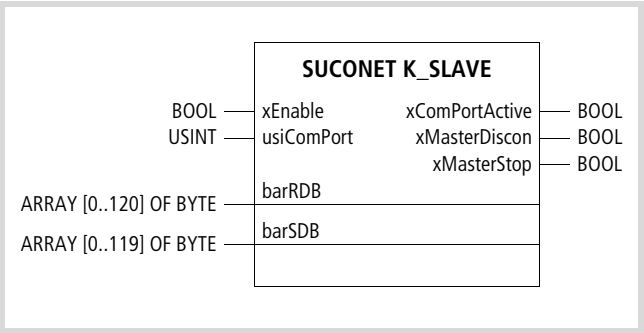
10 Communication block for Suconet K slaves: SuconetK.lib

Communication block

This function block is contained in the "SuconetK.lib" library.

→ This function block is for use with the XC100/XC200 in conjunction with the XIOC-SER module.

SUCONET K_SLAVE
Communication XC100/200 – Suconet K slave



Function block prototype

Meanings of the operands

xEnable	1 signal: Communication between CPU and XIOC-SER is enabled
	0 signal: Inhibit communication
usiComPort	Number of the COM interface: XC100: 2, 3 XC200: 2, 3, 4, 5
barRDB	Received data (max. 121 bytes: 120 data bytes and 1 diagnostics byte)
barSDB	Send data (max. 120 bytes)
xComPortActiv	1 signal: Communication between CPU and XIOC-SER is enabled
xMasterDiscon	1 signal: Suconet K-Master disconnected
xMasterStop	1 signal: Suconet K-Master on HALT

Description

The XIOC-SER module can be used as a Suconet K fieldbus connection for the XC100-CPU and XC200-CPU. The module operates as a slave on the Suconet K bus and is cyclically polled by the master; i. e. exchanges are made between the Slave CPU and the master data. This function block transmits and receives data to and from the user program.

Handling of the function block

Data transfer is enabled by a "1" signal on the "xEnable" input. If the parameters of the XIOC-SER module have been correctly set on the configurator (e. g. Bus status = SUCONET K), the "xComPortActiv" output is set to a "1" signal. Copy send data into the "barSDB" array, take received data from the "barRDB" array. Inspect the "xMasterDiscon" and "xMasterStop" outputs before every read-out: They should both have the "0" signal.

Apply a "0" signal to the "xEnable" input in order to stop data transmission. The "xComPortActiv" is then set to the "0" signal.

11 Communication block for Suconet K slaves: SuconetK.lib

The library contains the following function blocks:

- SuconetK_Master
- SuconetK_MSlaveData
- SuconetK_MDX2Data

General

→ These function blocks are for use with the XC100 and XC200. You also need an XIOC-NET-SK-M module.

With the XIOC-NET-SK-M module and the function blocks from the SuconetK_Master.lib you can easily and effectively replace older Suconet K or Suconet-K1 masters in existing applications.

You can connect up to 16 users to each module. Suconet K and Suconet K1 users are possible. The data volume per module is limited to 250 bytes send data and 250 bytes receive data, which includes the user's status data.

The XIOC-NET-SK-M module/COMport (slot, bus settings) can be made in the CoDeSys control configuration. It is described in "XIOC signal modules" (MN05002002Z-EN; previously AWB2725-1452GB) together with the interface assignment and bus termination resistors.

The bus stations (type, data volume) are configured in tables through a variable, which is usually global.

Description of the function blocks

Library SuconetK_Master.lib contains three function blocks. FB SuconetK_Master is always required. If you use it by itself, it performs a configuration of a Suconet K line and the data exchange between application and module.

The transmit/receive data to/from the slaves must be written to an array or read from an array.

Function blocks SuconetK_MSlaveData and SuconetK_MDX2Data are optional. They simplify programming access to the data for each user provided by FB SuconetK_Master.

FB SuconetK_MSlaveData, in combination with FB SuconetK_Master, simplifies programming standard slaves. FB SuconetK_MDX2Data in combination with FB SuconetK_Master is optimized for the EM4-201-DX2 with connected LEs.

SuconetK_Master

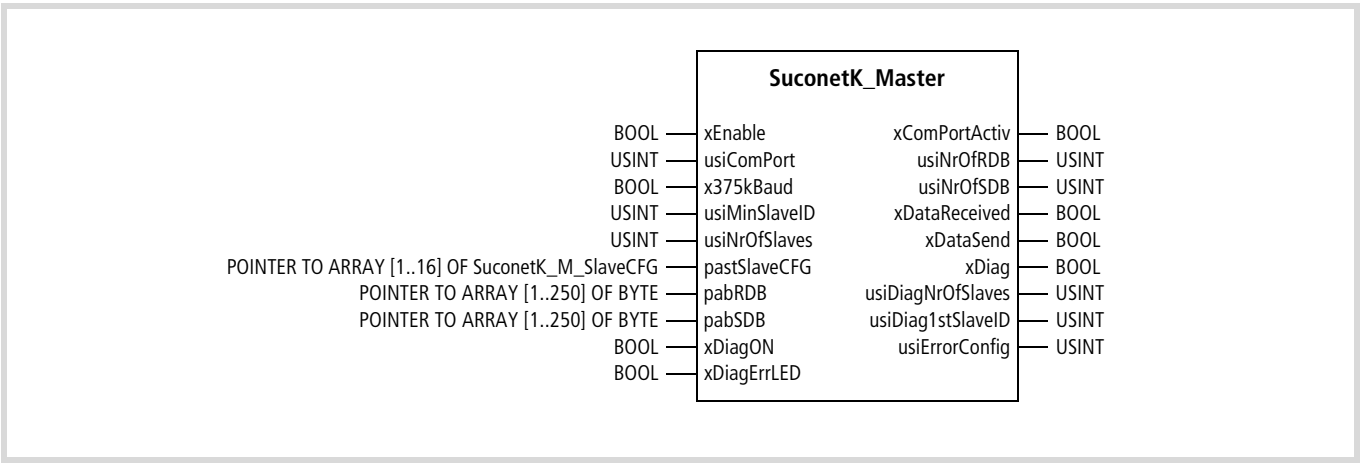


Figure 14: Function block prototype

Meanings of the operands

xEnable	Enable COM port, start communication
usiComPort	Number of COM port: XC100 → 2,3 XC200 → 2,3,4,5
x375kBaud	FALSE: 187.5 Kbit/s
	TRUE: 375 Kbit/s
usiMinSlaveID	Offset user address 1–30
usiNrOfSlaves	Number of connected bus users 1–16
pastSlaveCFG	Field address of slave configurations and slave information (one index per slave)
pabRDB	Address of the field with all slave read data
pabSDB	Address of the field with all slave write data
xDiagON	Diagnostics of Suconet run, TRUE: On, FALSE: Off
xDiagErrLED	Red error LED when diagnostics active

xComPortActiv	Status of COM port, Suconet K master active	
usiNrOfRDB	Number of read data of all configured users in byte; "255" for configuration error	
usiNrOfSDB	Number of write data of all configured users in byte; "255" for configuration error	
xDataReceived	Data received (pulse)	
xDataSend	Data sent (pulse)	
xDiag	Diag: one or more users are sending diagnostics information	
usiDiagNrOfSlaves	Diag: number of users with diagnostics information	
usiDiag1stSlaveID	Diag: address of first user with diagnostics information	
usiErrorConfig	Configuration error:	
	0	Configuration OK
	1..30	Number (SlaveID) of incorrectly configured user
	253	Baud rate 375 Kbit/s not possible with Suconet K1 user
	254	Error at usiMinSlaveId/usiNrOfSlaves
	255	Error when opening the COMports

SuconetK_MSlaveData

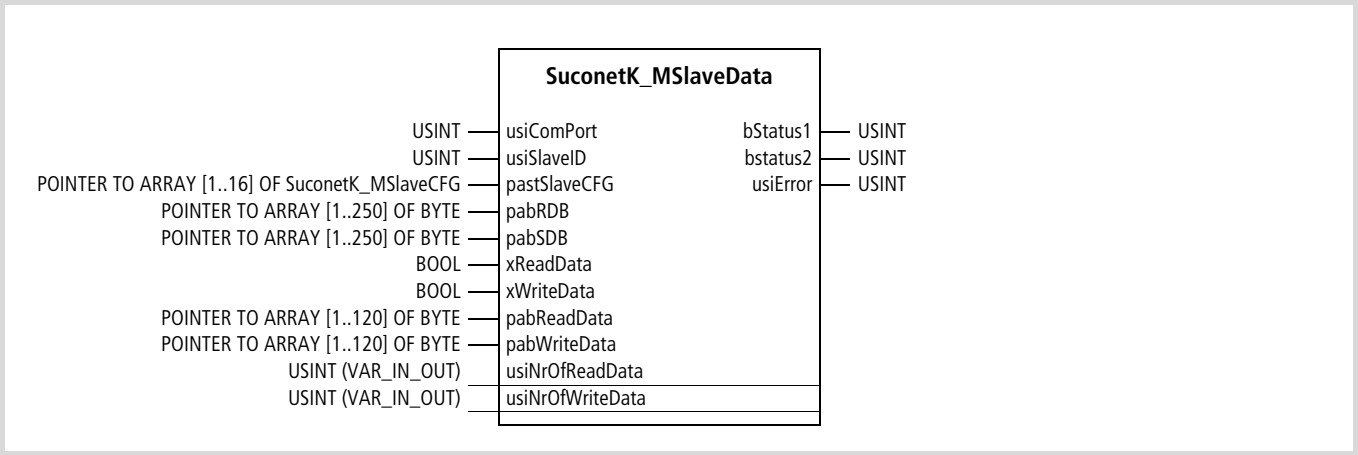


Figure 15: Function block prototype

Meanings of the operands

UsiComPort	Number of COMport: 2, 3, 4, 5
UsiSlaveID	Slave addresses 1..30
PastSlaveCFG	Field address of slave configurations and slave information (one index per slave)
PabRDB	Address of the field with all slave read data
PabSDB	Address of the field with all slave write data
XReadData	TRUE: read slave data from abRDB
XWriteData	TRUE: write slave data to abSDB
PabReadData	Address of the field for slave read data
PabWriteData	Address of the field for slave write data
UsiNrOfReadData	Number of bytes read or to be read
UsiNrOfWriteData	Number of bytes written or to be written
bStatus1	Suconet: user status byte
bStatus2	Device status byte (for EM4-101-AA2/TX1/TX2 only)
UsiError	Error code
	1 COMport not open
	2 SlaveID error
	3 Busbar run/COMport number error
	4 usiNrOfReadData error
	5 usiNrOfWriteData error
	6 usiNrOfReadData and usiNrOfWriteData error

SuconetK_MDX2Data for EM4-201-DX2

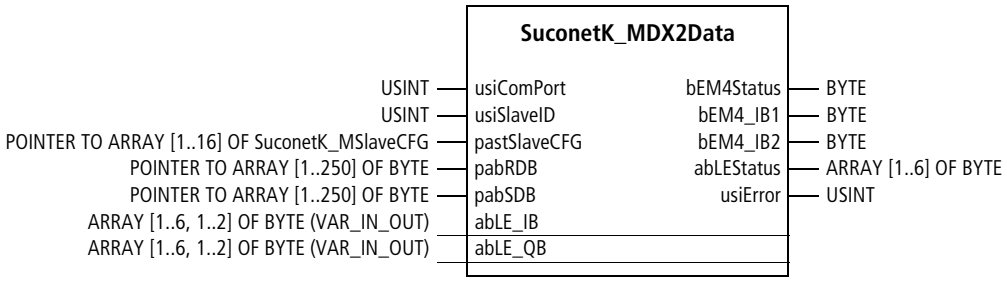


Figure 16: Function block prototype

Meanings of the operands

UsiComPort	Number of COMport: 2, 3, 4, 5
UsiSlaveID	Slave address 1–30 (EM4-201-DX2)
PastSlaveCFG	Field address of slave configurations and slave information (one index per slave)
PabRDB	Address of the field with all slave read data
PabSDB	Address of the field with all slave write data
abLE_IB	Input data of LE 1 to 6 at EM (max. 2 bytes/LE)
abLE_QB	Output data of LE 1 to 6 at EM (max. 2 bytes/LE)
bEM4Status	Suconet: EM4 status byte
bEM4_IB1	EM4: input byte 1
bEM4_IB2	EM4: input byte 2
AbLEStatus	Device status byte of LE 1 to 6
UsiError	Error code
	1 COMport not open 2 SlaveID error 3 Busbar run/COMport number error 4 Slave type error 5 Configuration error

Configuration of Suconet K line

To configure the Suconet K lines, create a variable – a global variable is best – of type ARRAY[1..x] of SuconetK_MSlaveCFG for each line, x being the number of users on this line. If you move the offset for the ID of the first user, the array indices also move accordingly.

The complete line is parameterized with the structural elements typSlave, xCRC, usiRDB, usiSDB, and typLE. Elements usiRDB and usiSDB are relevant only for users with configurable transmit and receive ranges, typLE is used only for configuring the local expansions for an EM4-201-DX2.

Once FB SuconetK_Master is enabled, it supplies structure element stInfo with further slave information (see section "Commissioning and operation of the Suconet K line").

Example configuration of a line with six users:

```
VAR_GLOBAL
SuconetK_COM2SlaveCFG : ARRAY[1..6] OF SuconetK_MSlaveCFG :=
    (* 1 *)    (typSlave:= XIOC_SER,      xCRC:=FALSE,  usiRDB:=120, usiSDB:=120),
    (* 2 *)    (typSlave:= PS4_201_MM1,   xCRC:=FALSE,  usiRDB:=20,usiSDB:=20),
    (* 3 *)    (typSlave:= EM4_201_DX2,   xCRC:=FALSE,  typLE:=LE4_116_XD1,LE4_108_XR1),
    (* 4 *)    (typSlave:= PS4_341_MM1,   xCRC:=FALSE,  usiRDB:=30,usiSDB:=30),
    (* 5 *)    (typSlave:= EM4_101_DD188, xCRC:=FALSE),
    (* 6 *)    (typSlave:= MI4_general,   xCRC:=FALSE,  usiRDB:=32,usiSDB:=32);
END_VAR
```

Important
The number of elements in this array must be at least the same as the number of users on the Suconet K bus (parameter usiNorOfSlaves on FB SuconetK_Master). Otherwise pointer access operations will cause runtime errors!

Also define the two arrays for the send and receive data of a line as global arrays:

```
VAR_GLOBAL
  SuconetK_COM2ReadData : ARRAY[1..250] OF BYTE;
  SuconetK_COM2WriteData : ARRAY[1..250] OF BYTE;
END_VAR
```

Commissioning and operation of the Suconet K line

Function block SuconetK_Master

For each line, declare and call an instance of FB SuconetK_Master. When COMport is enabled on xEnable, the configuration is verified. Any configuration errors are signalled immediately after the enable (usiErrorConfig). If the configuration is correct, it is transferred to the module. SuconetK_MSlaveCFG.stInfo provides the application with information about the slave data: offset, number of status and receive data bytes within array abRDB, and offset and number of possible send data bytes array abSDB. Parameters usiNrOfRDB and usiNrOfSDB contains the number of used receive and send data bytes after a successful module start.

Example configuration of FB SuconetK_Master

```
SuconetK_MasterCOM2(
  xEnable:=COM2enable ,
  usiComPort:=2 ,
  x375kBaud:=FALSE,
  usiMinSlaveID:=SuconetK_COM2usiMinSlaveID,
  usiNrOfSlaves:=SuconetK_COM2usiMaxSlaveID,
  pastSlaveCFG:=ADR(SuconetK_COM2SlaveCFG),
  pabRDB:=ADR(SuconetK_COM2ReadData),
  pabSDB:=ADR(SuconetK_COM2WriteData),
  xDiagON:=COM2DiagON,
  xDiagErrLED:=COM2DiagLED,
  xComPortActiv=>COM2activ ,
  usiNrOfRDB=>COM2RDBtotal,
  usiNrOfSDB=>COM2SDBtotal,
  xDataReceived=>COM2RecData ,
  xDataSend=>COM2SendData ,
  xDiag=>COM2Diag ,
  usiDiagNrOfSlaves=>COM2DiagSlaves ,
  usiDiag1stSlaveID=>COM2DiagSlaveID ,
  usiErrorConfig=> COM2ConfigError );
```

Calling up FB SuconetK_Master once per Suconet K line configures XIOC-NET-SK-M and performs the cyclic data exchange between module and application during operation. Each time it is called, the FB writes the receive data of all users – including status bytes – to array abRDB and passes the send data from array abSDB to the module. The data offsets and number for each user can be read from the SuconetK_MSlaveCFG.stInfo structure. xDataReceived and xDataSend signal the corresponding data refresh.

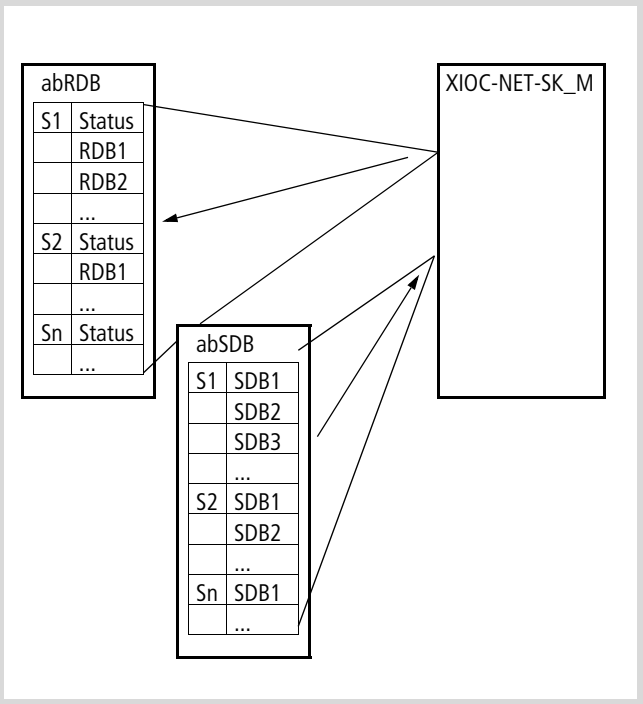


Figure 17: Data ranges of FB Suconet K Master

Diagnostics during operation

With input xDiagON on FB SuconetK_Master, this FB polls all users with each call and signals problems through output xDiag, usiNiOfSlaves and usiDiag1stSlaveID (ERR-LED if xDiagErrLED = TRUE). For xDiagON, the status bytes are output and updated under SuconetK_MSlaveCFG.stInfo.stDiag.

Table 7: Diagnostic LEDs

POW	Power supply
ERR	SuconetK_Master → with xDiagON only
DTR	COMport open and module configured
DCD	All configured users OK
TxD	Data is being sent to user
RxD	Data is being received by user

Function block SuconetK_MSlaveData

In connection with FB SuconetK_Master, this FB provides convenient access to the data of a user (including EM4-201-DX2 with connected LEs). Depending on the parameterization of xReadData and xWriteData, it reads and/or writes the number usiNrOfReadData/usiNrOfWriteData of data packages from abReadData/abWriteData from or to the user.

If this variable has a value of zero, the configured RDBs/SDBs or – if the data length is fixed – the permissible number of data bytes are automatically transferred.

Important
The number of data bytes must not exceed the size of the corresponding data arrays (pointer access). If only one transfer direction is selected, you do not have to parameterize the other data array.

The user status bytes are updated at every call, including xReadData and xWriteData = FALSE, regardless of whether SuconetK_Master diagnostics is enabled or not.

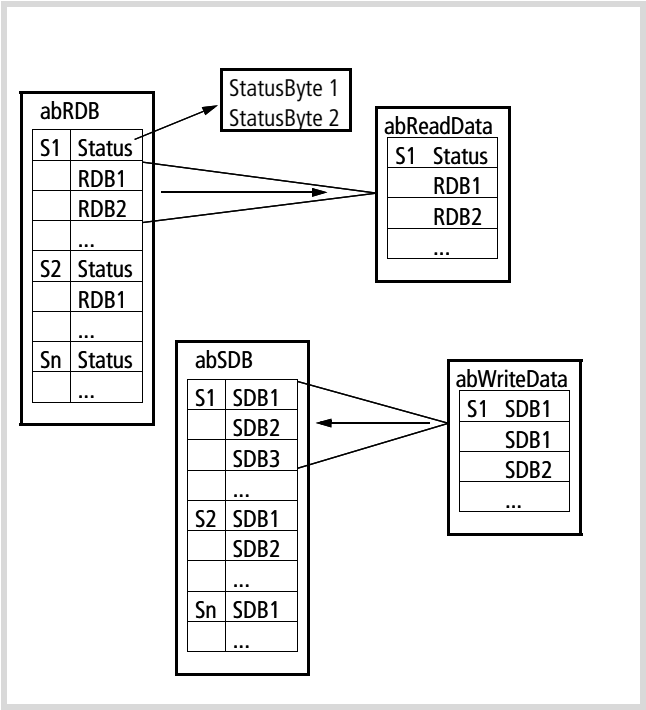


Figure 18: Data ranges of FB Suconet K MSlaveData

Function block SuconetK_MDX2Data

Like FB SuconetK_MslaveData, this FB also transmits data from/to abRDB/abSDB.

It provides improved access to the data from data an EM4-201-DX2 and the connected LEs. While FB SuconetK_Master and SuconetK_MSlaveData attach the LEs' status and data bytes to the data from the EM4-201-DX2 in one block, SuconetK_MDX2Data assigns the data bytes to individual array segments.

Type definitions of the library

Configuration of a Suconet user

```
TYPE SuconetK_MSlaveCFG :
  STRUCT
    typSlave:SuconetK_MSlaveType := noSlave;
    xCRC:BOOL := FALSE;
    usiRDB: USINT := 0;
    usiSDB:USINT := 0;
        typLE:ARRAY[1..6] OF SuconetK_MLEType :=
        noLE,noLE,noLE,noLE,noLE,noLE;
    stInfo:SuconetK_MSlaveInfo;
  END_STRUCT
END_TYPE
```

Input : Possible slave types

TYPE SuconetK_MSlaveType:

```
( noSlave:=0,
  A4_220,          LE4_501_BS1,    PS4_401_K,     SIS_TYP_80DF,
  A5_220,          MI4_general,    PS4_401_K1,    SIS_TYP_80E0,
  CM4_501_FS1,     MI4_101_KC1,    RBI1,          SIS_TYP_80E1,
  CM4_504_GS1,     MI4_101_KE1,    RMQ_16I,       SIS_TYP_80E2,
  CM4_505_GS1,     MI4_111_KE1,    RMQ_16I_K,     SIS_TYP_80E3,
  DE4_NET_K,       MI4_131_KH1,    SBI_AMD3,      SIS_TYP_80E4,
  DE4_NET_K_F,     MI4_151_KF1,    SBI_AMX,       SIS_TYP_80E5,
  EBE295,          MI4_151_TA1,    SIS_K_06_07,   SIS_TYP_80E6,
  EM4_101_AA1,     MI4_161_TC1,    SIS_K_10_10,   SIS_TYP_80E7,
  EM4_101_AA1B63,  MI4_451_KF1,    SIS_K_15_15,   SIS_TYP_80E8,
  EM4_101_AA1B64,  MI4_451_TA1,    SIS_K_24_24,   SIS_TYP_80E9,
  EM4_101_AA1W31,  MI4_471_TC1,    SIS_K_30_30,   SIS_TYP_80EA,
  EM4_101_AA1W33,  MV4,            SIS_K_40_40,   SIS_TYP_80EB,
  EM4_101_AA2B84,  PS306,          SIS_K_50_50,   SIS_TYP_80EC,
  EM4_101_AA2W84,  PS3_8,          SIS_K_60_60,   SIS_TYP_80ED,
  EM4_101_DD1106,  PS3_AC,         SIS_TYP_80D0,  SIS_TYP_80EE,
  EM4_101_DD188,   PS3_DC,         SIS_TYP_80D1,  SIS_TYP_80EF,
  EM4_101_DD2106,  PS4_101_DD1,    SIS_TYP_80D2,  SIS_TYP_A0EF,
  EM4_101_DD288,   PS4_111_DR1,    SIS_TYP_80D3,  VTP0_H_T1_K1,
  EM4_101_TX1,     PS4_141_MM1,    SIS_TYP_80D4,  VTPx_H_T6_K,
  EM4_101_TX2,     PS4_151_MM1,    SIS_TYP_80D5,  XIOC_SER,
  EM4_111_DR1,     PS416_CPU_300,  SIS_TYP_80D6,  ZB4_501_TC1,
  EM4_111_DR2,     PS416_CPU_400,  SIS_TYP_80D7,  ZB4_501_TC2,
  EM4_201_DX1,     PS416_NET_400,  SIS_TYP_80DA,  ZB4_501_UM2,
  EM4_201_DX2,     PS4_201_MM1,    SIS_TYP_80DB,  ZB4_501_UM3,
  EPC335,          PS4_271_MM1,    SIS_TYP_80DD,  ZB4_501_UM4);
  EPC335_K,        PS4_341_MM1,    SIS_TYP_80DE,
END_TYPE
```

Input : Possible LE types (for EM4_201_DX2 only)

```
TYPE SuconetK_MLEType :
( noLE := 0,
  LE4_104_XP1,
  LE4_108_XD1,
  LE4_108_XR1,
  LE4_116_DD1,
  LE4_116_DX1,
  LE4_116_XD1,
  LE4_308_HX1,
  LE4_308_XH1);
END_TYPE
```

Output : Information about the address ranges of the configured slaves

```
TYPE SuconetK_MSlaveInfo :
  STRUCT
    usiSlaveID:USINT;
    usiStatusOff:USINT;
    usiStatusNr:USINT;
    usiReadOff:USINT;
    usiReadNr:USINT;
    usiWriteOff:USINT;
    usiWriteNr:USINT;
    stDiag:SuconetK_MDiagStatus;
  END_STRUCT
END_TYPE
```

Output : Status bytes for Diag (SuconetK_Master)

```
TYPE SuconetK_MDiagStatus :
  STRUCT
    typStatus1 : SuconetK_MStatusByte1;
    bStatus2 : BYTE;
  END_STRUCT
END_TYPE
```

12 Diagnostics function blocks: XSysDiagLib.lib

Two function blocks are available for analyzing the diagnostic data:

- xDiag_SystemDiag → page 89
- xDiag_ModuleDiag → page 90

With the Xdiag_GetSupplierVersion function you can read out the version of library XSysDiagLib.lib, → page 88.

→ The function/function blocks can be used for XC100/XC200 and XN-PLC-CANopen.

→ This FB can not be used for diagnostics on the CANopen bus at the CANopen–CPU interface.

Software requirements:

- XC100: from V3.10
- XC200: from V1.03.02

Diagnostics overview

With the diagnostics function blocks you can read the diagnostic data of the following users:

- XI/ON modules on the XN-PLC
- Slaves on the PROFIBUS-DP line (→ note below)
- DP-S module

→ Two methods are available for reading diagnostic data from DP slaves:

  – For existing applications use function blocks GETBUSSTATE and DIAGGETSTATE. The procedure is described in the manual "XIOC signal modules" (MN05002002Z-EN; previously AWB2725-1452GB).
  – For new applications use function blocks "xDiag_SystemDiag" and "xDiag_ModuleDiag".

The diagnostics are split into module diagnostics and specific diagnostics. Module diagnostics can be performed with PLCs XC100, XC200 and XN-PLC. The result is a concise listing of the diagnostic data of all diagnostic-capable modules or PROFIBUS-DP users. The specific diagnostics depends on the modules. The result is a detailed listing of the diagnostic data of all diagnostic-capable modules or PROFIBUS-DP users.

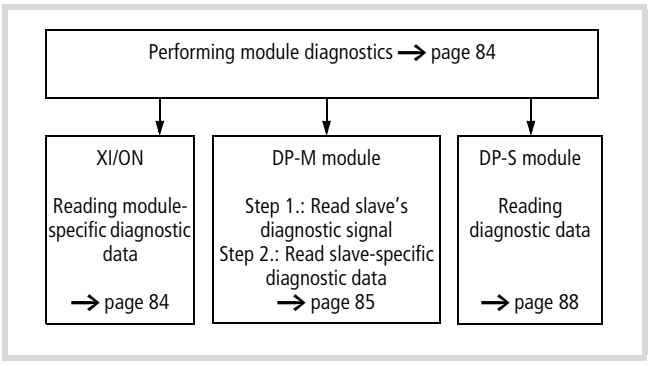


Figure 19: Module information display

Performing module diagnostics

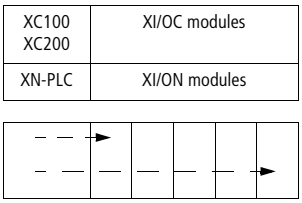


Figure 20: Module diagnostics

For module diagnostics, call function block "xDiag_SystemDiag". Output "abyModuleInfo" (array of byte) contains the modules' diagnostic data. Scan the array's individual elements (diagnostic bytes). The first element contains the diagnostic data for the first module, etc.

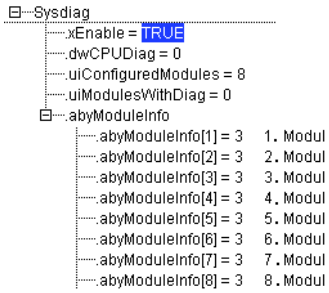


Figure 21: Displaying module information

You can evaluate bits 0, 1 and 2 from the diagnostic byte. Diagnostic byte "abyModuleInfo", for example, has a value of 3 in figure 21, which corresponds with the value 0000 0011_{bin} and means:

Bit 0 = 1:	Module configured[1)]
Bit 1 = 1:	Data exchange OK[1)]

1) For details about the diagnostic byte, see section "xDIAG_SystemDiag" on page 89.

The following sections describes how to determine module-specific diagnostic data.

Module-specific diagnostic data of XI/ON modules

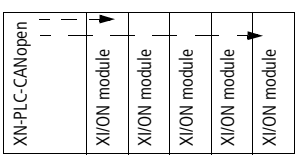


Figure 22: Diagnosing XI/ON modules

→ Some XI/ON modules – for example digital input modules – can not send module-specific diagnostic data to the XN-PLC. For details, read the documentation for the XI/ON modules.

If bit 2 from diagnostic byte "abyModuleInfo" of an XI/ON module is set, you can read the module's specific diagnostic data through function block "xDiag_ModuleDiag". To do this, enter the slot number at input "uiSlot" and the module's slot at "uiIndex" and start the function block. The diagnostic data then display output array "abyExtendedInfo".

Example
Output module 2DO-24VDC-0.5A-P indicates a short-circuit.

When you call function block "xDiag_ModuleDiag" and output array "abyExtendedInfo" in the status indication, the following view appears:

(The module is on slot (uiSlot) = 5)

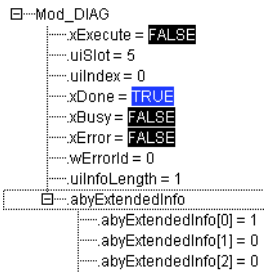


Figure 23: Short-circuit indication of XI/ON output module

The "1" in byte "abyExtendedInfo[0]" indicates the short-circuit.

Reading diagnostic data from slaves on the DP line

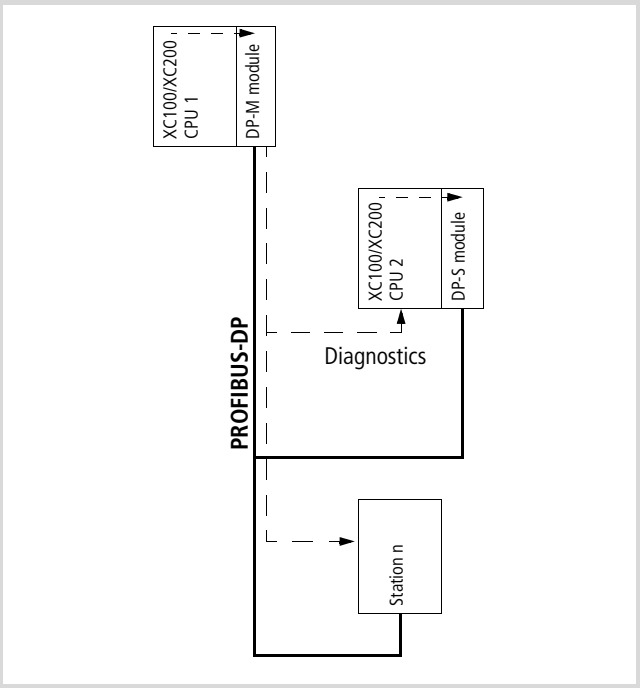


Figure 24: Diagnostics on the PROFIBUS-DP line

The query consists of two steps, which are described in more detail below:

- Reading the status bytes of all slaves. The status bytes indicate whether a diagnosis was performed.
- Reading the slave-specific diagnostic data of the slaves that reported a diagnosis.

Reading the status bytes of all slaves.

The general scan returns a list (array) of all slaves' status bytes.

Precondition:
Bit 2 of the DP-M module's diagnostic byte "abyModuleInfo" is set, i.e. diagnostic data exist.

Procedure:
Call function block "xDiag_ModuleDiag". To do this, enter the slot number at the "uiSlot" inputs and a zero at "uiIndex" and start the function block.

Output array "abyExtendedInfo" shows the status bytes of each slave. Each slave has a status byte.

→ Output array "abyExtendedInfo", which is described here, corresponds with output array "ExtendedInfo" of the type GETBUSSTATE variables.
See manual "XIOC Signal Modules" (MN05002002Z-EN; previously AWB2725-1452GB).

Table 8: Status byte

Bit	7	6	5	4	3	2	1	0	Slave address
Byte 0									
Byte 1									
Byte 2						×	×	×	2
Byte 3						×	×	×	3
...									
Byte 125						×	×	×	125

Scan bits 0, 1 and 2 of each of the slaves' status bytes to check for diagnostic messages, starting with address 2 and up to 124.

Table 9: Meaning of bits 0, 1 and 2 of the status byte

Bit 0	= 1:	A configuration exists for this address
Bit 1	= 1:	Data exchange OK[1)]
Bit 2	= 1:	New diagnostic data exists

1) Bit 1 already indicates a "1" signal when data exchange for coupling of the slave has been successful. This means that the connection is OK and data exchange occurs.

Example:
In figure 25 slave 2 has a value of 5_{dez}, which corresponds with 00000101_{bin} and has the following meaning:

Bit 0 = 1: A configuration exists
bit 1 = 0: The data exchange is interrupted
Bit 2 = 1: Diagnostic data exists for the second user.

The next section describes the procedure for reading this diagnostic data for user 2.

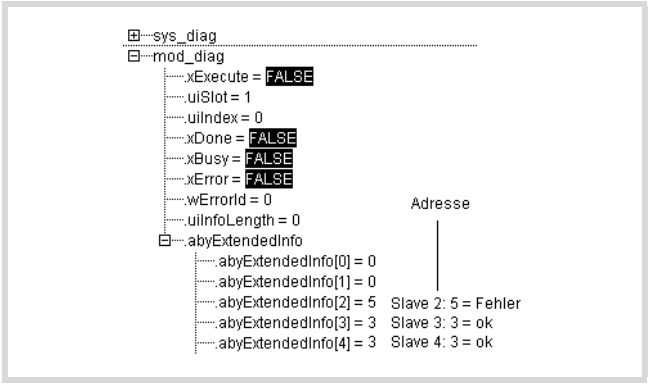


Figure 25: Overview of status byte of slaves 2 to 4 (status indication)

Reading the slave-specific diagnostic data

Call function block "xDiag_ModuleDiag", enter the module's slot number at the "uiSlot" inputs and the slave number at "uiIndex". Start the function block. Output array "abyExtendedInfo" returns the user's slave-specific diagnostic data. It is arranged into:

- General diagnostics data (Byte 0 to 7)
- Standard diagnostic data (bytes 8 to 13) to DP standard.
- Device-specific diagnostic data (bytes 14 to 99)
  → device documentation and associated GSD file.

The most important data has a grey background in the table below.

EXTENDEDINFO[0]	//with PROFIBUS-DP: slave address		
EXTENDEDINFO[1..4]	//no meaning		
EXTENDEDINFO[5]	//length byte of the device diagnostic		
EXTENDEDINFO[6&7]	//no meaning		
EXTENDEDINFO[8..13]	//6 octet DP standard diagnostic conform to standard		
EXTENDEDINFO[8] (Standard byte 1)	//Status_1		
	Bit 0:	Device does not respond (no valid I/O data)	
	Bit 1:	Slave not ready	
	Bit 2:	Divergent configuration	
	Bit 3:	Extended diagnostics data present	
	Bit 4:	Unknown command	
	Bit 5:	Invalid response	
	Bit 6:	Incomplete parametric programming	
	Bit 7:	Parametric programming from another master	
EXTENDEDINFO[9] (Standard byte 2)	//Status_2		
	Bit 0:	Ready for new starting sequence	
	Bit 1:	No parametric programming	
	Bit 2:	"1"	
	Bit 3:	Watchdog activated	
	Bit 4:	FREEZE command active	
	Bit 5:	SYNC command active	
	Bit 6:	Reserved	
	Bit 7:	Slave has not been engineered	
EXTENDEDINFO[10] (Standard byte 3)	//no meaning		
EXTENDEDINFO[11] (Standard byte 4)	//for PROFIBUS-DP: master address		
EXTENDEDINFO[12&13] (Standard byte 5, 6)	//Own identity number		
EXTENDEDINFO[14]	// length byte of device-specific data		
EXTENDEDINFO[15..99]	//device-specific diagnostics.		

Diagnostics capable XI/ON modules

If you perform diagnostics with the DIAGGETSTATE function block on an XI/ON station, the EXTENDEDINFO output displays the diagnostics data for the entire station in bytes 15 and 16. The data originate from the GSD file of the central XI/ON gateway.

Byte 17 to 99 contains the fault code for the modules with diagnostics capability. This occurs in the module sequence. A byte will not exist for non-diagnostic capable modules.

EXTENDEDINFO[15]	//	Bit 0:	Module diagnostics present
		Bit 2:	Parametric programming incomplete
		Bit 3:	Divergent configuration
EXTENDEDINFO[16]	//	Bit 1:	–
		Bit 2:	Module bus fault
		Bit 3:	Master configuration fault
		Bit 4:	–
		Bit 5:	Station configuration fault
		Bit 6:	I/Oassistant force mode active
		Bit 7:	Module bus failure
EXTENDEDINFO[17...99]	→ table below. For further information, see also manual "XI/ON PROFIBUS-DP" (AWB2700-1394G).		

The following excerpt from the "XI/ON Gateways for PROFIBUS-DP" (MN05002004Z-EN; previously AWB2725-1529G) manual indicates the diagnostics bit of the XI/ON modules:

e.g. power supply module		
XN-BR-24VDC-D	Bit 0:	Module bus voltage warning
	Bit 2:	Field voltage missing
XN-PF-24VDC-D	Bit 2:	Field voltage missing
XN-PF-120/230VAC-D	Bit 2:	Field voltage missing
e.g. output modules		
XN-2DO-24VDC-0.5A-P XN-2DO-24VDC-2A-P XN-2DO-24VDC-0.5A-N XN-16DO-24VDC-0.5A-P	Bit 0:	Overcurrent channel 1
	Bit 1:	Overcurrent channel 2

e.g. analog module		
XN-1AI-I	Bit 0:	Measured value range fault
	Bit 1:	Wire breakage
XN-1AI-U	Bit 0:	Measured value range fault
XN-2AI-PT/NI-2/3	1st BYTE	
	Bit 0:	Measured value range fault (channel 1)
	Bit 1:	Wire breakage
	Bit 2:	Short-circuit
	2nd BYTE	
	Bit 0:	Measured value range fault (channel 2)
	Bit 1:	Wire breakage
	Bit 2:	Short-circuit
e.g. counter module		
XN-1CNT-24VDC (C)	Bit 0:	Short-circuit/wire breakage DO
	Bit 1:	Short-circuit 24 V DC encoder supply
	Bit 2:	Count range end false
	Bit 3:	Count range start false
	Bit 4:	Invert DI with L ret. fault
	Bit 5:	Main count direction false
	Bit 6:	Operating mode false
XN-1CNT-24VDC (M)	Bit 0:	Short-circuit/wire breakage DO
	Bit 1:	Short-circuit 24 V DC encoder supply
	Bit 2:	Encoder impulse false
	Bit 3:	Integration time false
	Bit 4:	Upper limit false
	Bit 5:	Lower limit false
	Bit 6:	Operating mode false
e.g. DOL starter module		
XS1-XBM	Bit 0:	Ident fault
	Bit 1:	PKZ short-circuit
	Bit 2:	PKZ overload
	Bit 4:	DIL1 defective
	Bit 5:	DIL2 defective

Example
A slave with address 2 can not be linked.

When you call function block "xDiag_ModuleDiag" and output array "abyExtendedInfo" in the status indication, the following view appears:

(The DP-M module is on slot (uiSlot) = 1)

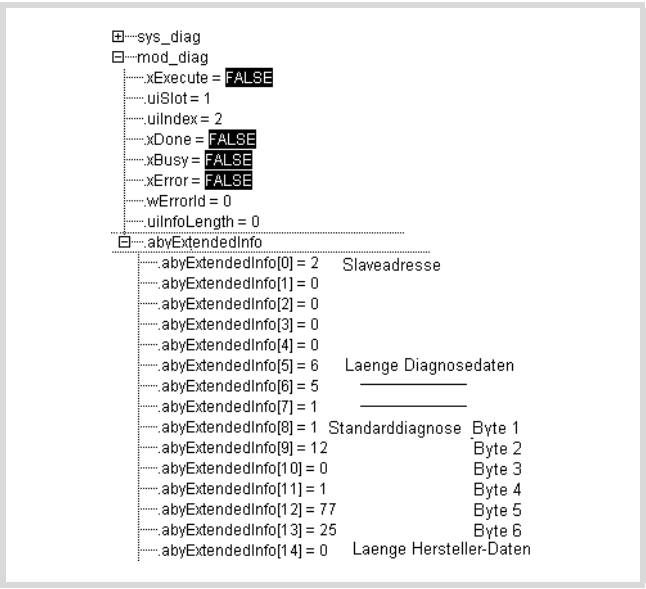


Figure 26: Slave-specific diagnostic data of the slave (address 2), → table 10

Table 10:Evaluation of standard diagnostics

Standard diagnostics byte n AbyExtendedInfo[...]	**Value**	**Meaning**
Standard diagnostics byte 1 AbyExtendedInfo[8]	1	Slave not ready
Standard diagnostics byte 2 AbyExtendedInfo[9]	12	Watchdog enabled
Standard diagnostics byte 3 AbyExtendedInfo[10]	0	No function
Standard diagnostics byte 4 AbyExtendedInfo[11]	1	Master address 1
Standard diagnostics byte 5 AbyExtendedInfo[12]	77	ID number
Standard diagnostics byte 6 AbyExtendedInfo[13]	25	ID number

Diagnostic data of the DP-S module

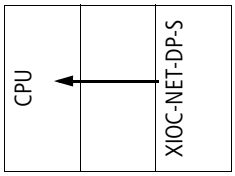


Figure 27: Diagnostic data of a DP-S module

To determine whether the DP-S module is still communicating with the master, call function block "xDIAG_SystemDiag" and scan the diagnostic byte of the DP-S module at output array "abyModuleInfo".

If bit 2 of the DP-S module's diagnostic byte is set, you can read the module's diagnostic data. To do this, call function block "xDiag_ModuleDiag", enter the module's slot number at the "uiSlot" inputs and a zero at "uiIndex" and start the function block. The listed bytes of output array "abyExtendedInfo" show the communication status:

AbyExtendedInfo[1] =	16_{dec}	Data exchange with DP master OK,
	0_{dec}	not OK
AbyExtendedInfo[2, 3]		Number of configured data blocks (bytes) to be sent
AbyExtendedInfo[4, 5]		Number of configured data blocks (bytes) to be received

Compatibility

You can use the usually procedure for diagnosing PROFIBUS-DP lines (XIOC-NET-DP-M) (⟶ MN05002002Z-EN; previously AWB2725-1452GB, section "XIOC-NET-DP-M"). To do this, set bit "EnableDiags" to "Yes" under Controller configuration → Module parameters.

Alternatively, you can use the function blocks "xDiagSystemDiag" and "xDiagModuleDiag" described here. In that case, set bit "EnableDiags" to "No" (default setting).

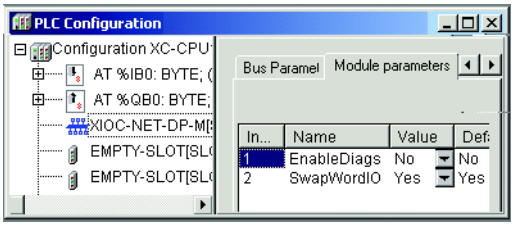


Figure 28: Enabling marker range for diagnostics

Reading the version

With function "xdiag_GetSupplierVersion" you can read the version of library XSysDiagLib.lib. The first version returns the value 16#0101.

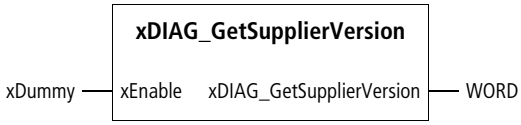


Figure 29: Function block prototype

"xDIAG_SystemDiag"

This function block ready system diagnostic data.

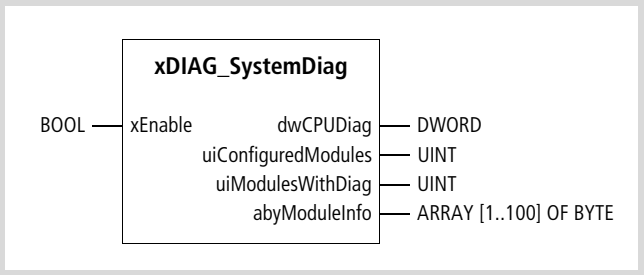


Figure 30: Function block prototype

→ Under "Global variables", declare function block "xDiag_SystemDiag".

Meanings of the operands

xEnable	As long as the input is TRUE, the function block's outputs are updated each time the function block is called. When xEnable is FALSE, all FB outputs are 0.
dwCPUDiag	Not yet defied!
uiConfigured Modules	This output returns the number of configured XI/OC or XI/ON modules.
uiModulesWith Diag	This output returns the number of modules that are currently reporting module-specific diagnostic data.
abyModuleInfo	This array holds a diagnostic byte for each module, with the following information, → also table 11:

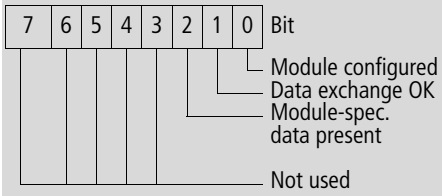


Table 11:Meaning of bits 0, 1 and 2 of the diagnostic byte

		Applies to all XI/OC and XI/ON modules	
Bit 0	= 1:	Module configured An XI/OC or XI/ON module is configured for the slot. The bit does not specify whether a module is physically present or whether the present module corresponds with the configured module type.	
	= 0:	Not configured	
Bit 1	= 1:	Data exchange OK The slot contains a module and the module type corresponds with the configuration.	
	= 0:	No data exchange (configuration fault)	
		Valid for XI/ON and DP-M	Valid for DP-S
Bit 2	= 1:	Diagnostic data exist	Connection between DP-S and DP-M faulty
	= 0:	No diagnostic data	Connection OK

PLC XC100 and XC200 determine the status of bits 1 and 2 once at startup after a full reset and when the power supply is switched on. Failure and removal of an I/O module during operation are not detected by the PLCs. The XN-PLC also polls the XI/ON modules during RUN operation and sets bits 0 to 2.

xDIAG_ModuleDiag

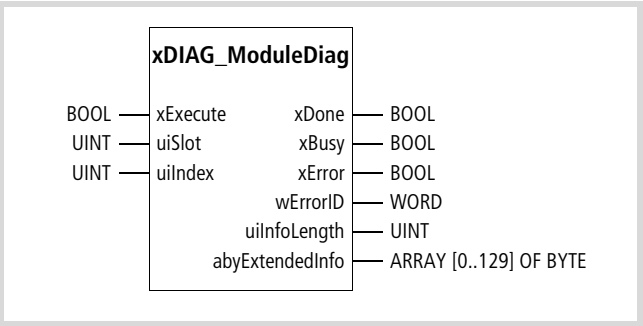


Figure 31: Function block prototype

Description

This function block reads the module-specific diagnostic data of modules XIOC-NET-DP-M and XIOC-NET-DP-S in the XC100/XC200 and of an XI/ON module in the XN-PLC-CANopen. You can also use this function block to diagnose the users of a PROFIBUS-DP line.

Meanings of the operands

xExecute	A positive edge at the input starts a once-only execution of the function block.
uiSlot	The input specifies the slot for which diagnostic data is to be supplied. The following rule applies: uiSlot = 1 – 15 : Diagnostics of XI/OC slot 1 – 15 (XC100/XC200) uiSlot = 1 – 75 : Diagnostics of XI/ON slot 1 – 75 (XN-PLC-CANopen)
uiIndex	This input is only for use in connection with a DP-M module. Specify the slot of the DP-M module at input "uiSlot". • Uindex = 0: Output array "abyExtendedInfo" contains the DP users' status bytes. • Uindex = 1...124 ()user address: Output array "abyExtendedInfo" contains DP user diagnostic data Address 0 is not allowed. DP-S module: Uindex = 0
xDone	The output TRUE when the function block has finished. When it is restarted, it changes to FALSE. The function block output does not provide information about whether execution has been completed without errors.
xBusy	The output is: • TRUE during FB execution • FALSE after FB execution.
xError	The output provides the following information after FB execution: xError = FALSE: OK xError = TRUE: error, error code → "uiErrorId" When the FB is restarted, "xError" changes to FALSE.
uiErrorId	Error code: 0: No error 1: Invalid slot 2: Invalid index 3: No diagnostic data present 4: Slave not accessible (PROFIBUS-DP only) 5: Slave sending error response (PROFIBUS-DP only)*1) *1) When this error occurs, the DP slave's corresponding error code is written to abyExtendedInfo[0], and uiInfoLength supplies the value 1. When the FB is restarted, the output changes to 0.
uiInfoLength	This output returns the length of the supplied, module-specific diagnostic data.
abyExtendedInfo	This array contains the diagnostic data for the slave specified with "uiIndex".

Diagnostics example

This sample program diagnoses n slaves (n = DP_MAX_SLAVE_ADR) on a PROFIBUS-DP line. Function blocks "xDIAG_SystemDiag" and "xDIAG_ModuleDiag" from library "xSysDiagLib.lib" are used. You can adapt this example program to a specific application.

The following devices are available:

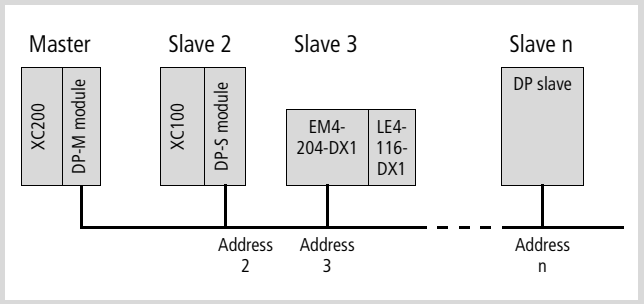


Figure 32: Device configuration for the diagnostics example

Program description

In section "Call system diagnostics", function block "xDIAG_SystemDiag" (xiocSysDiag) is started. Program section "Determine whether master has diagnostic messages" scans abyModuleInfo[1] (1= DP master's slot number). If bit 2 is set, function block "xDIAG_ModuleDiag" (dpBusState) is called.

In program section **** Determine which slave is reporting diagnostics ***** the slave's bytes are scanned for the presence of diagnostics data. If diagnostics data is present, function block "xDIAG_ModuleDiag" is started with "dpmSlaveDiag" for the specific slave.

The module diagnostics are then evaluated. In array "iDPDiagCnt", the diagnostic messages for each slave are counted. Array "iDpDiagCnt" in the status indication has the following structure:

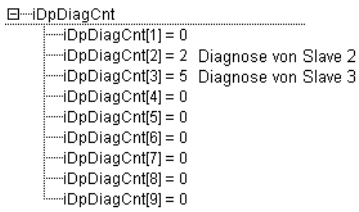


Figure 33: Number of diagnostic messages

```
FUNCTION_BLOCK DP_Sys_Modul_Diag_Master

VAR_INPUT
     DP_SLOT_NR:INT; (*:=1;*)                        (* DP Master is in slot 1 *)
     DP_MAX_SLAVE_ADR:INT; (*:=124;*)                (* Highest slave address *)
END_VAR

VAR_OUTPUT
     FieldDiag_DP: ARRAY[0..129] OF BYTE;            (* Additional field for diagnostic data (optional) *)
END_VAR

VAR
     xiocSysDiag: xDIAG_SystemDiag;                  (* Diagnostics of XI/OC.nodes *)
     dpBusState: xDIAG_ModuleDiag;                   (* Determine which slaves are reporting diagnostics *)
     dpmSlaveDiag: xDIAG_ModuleDiag;                 (* Read slaves' diagnostic data *)
     iDpDiagCallCnt : ARRAY [1..124] OF INT:=0;      (* Show number of times each slave reports diagnostics *)
     uiSlaveDiagErrCnt: UINT:=0;
     i:INT:=1;
     xDiagAvailable: BOOL;
     bDiagInLastCyc: BYTE;
END_VAR
```

```
(****************************** Call system diagnostics ******************************************)

IF NOT xiocSysDiag.xEnable THEN
    xiocSysDiag.xEnable:=TRUE;
END_IF
xiocSysDiag();

(********************** Determine whether master has diagnostic messages *********************)

(* If a system diagnostic is present in the DP-Master
[or in the last cycle (change from Diag to not Diag)],
then query the DP bus status *)
IF(( xiocSysDiag.abyModuleInfo[DP_SLOT_NR] AND 16#04) = 04 OR bDiagInLastCyc>0) THEN
    IF NOT dpBusState.xBusy AND NOT dpBusState.xExecute THEN
           dpBusState.uiSlot:=DP_SLOT_NR;
           dpBusState.uiIndex:=0;
           dpBusState.xExecute:=TRUE;
    END_IF
    dpBusState();
    IF dpBusState.xDone THEN
           dpBusState.xExecute:=FALSE;
           dpBusState();
           bDiagInLastCyc:=xiocSysDiag.abyModuleInfo[DP_SLOT_NR] AND 16#04;
    END_IF
END_IF

(********************** Determine which slave is reporting diagnostics *********************************)

(* If no Diag FB is active, find next slave with diagnostics *)
IF (NOT xDiagAvailable) THEN
    WHILE (i<>(DP_MAX_SLAVE_ADR+1))
(* DP configuration up to slave address "DP_MAX_SLAVE_ADR"*)
    DO
           i:=i+1;              (* Zero is not allowed! *)
(* Query which slave is reporting diagnostics data; bit 2 set *)
           IF (dpBusState.abyExtendedInfo[i] AND 16#04) = 04 THEN
                                xDiagAvailable:=TRUE;
                                EXIT;               (* Exit loop when a slave reports diagnostics
           END_IF
    END_WHILE
                                                    (* If no slave found, start from the top *)
    IF i > DP_MAX_SLAVE_ADR THEN
           i:=0;
           xDiagAvailable:=FALSE;
    END_IF
END_IF
```

```
(********************* Read slave diagnostic data ********************************)

IF (xDiagAvailable) THEN                            (* Slave diagnostics data present *)
    IF NOT dpmSlaveDiag.xBusy AND NOT dpmSlaveDiag.xExecute THEN
          dpmSlaveDiag.uiSlot:=DP_SLOT_NR;
          dpmSlaveDiag.uiIndex:=i;                  (* Slave address from search loop *)
          dpmSlaveDiag.xExecute:=TRUE;
    END_IF
    dpmSlaveDiag();
    IF dpmSlaveDiag.xDone THEN
          IF dpmSlaveDiag.xError THEN               (* Read error message*)
                              uiSlaveDiagErrCnt:=uiSlaveDiagErrCnt+1;
          ELSE                                      (* Count received diagnostic messages *)
                              iDpDiagCnt[i]:=iDpDiagCnt[i]+1;
          FeldDiag_DP:=dpmSlaveDiag.abyExtendedInfo;

          END_IF
          dpmSlaveDiag.xExecute:=FALSE;
          dpmSlaveDiag();
          xDiagAvailable:=FALSE;
    END_IF
END_IF
```

13 Transparent mode functions: xSysCom200/SysLibCom/XC100_SysLibCom/XN-PLC-SysLibCom.lib

These libraries are for use with the CPUs and the XIOC-SER modules.

Library	XC200	XC100	XN-PLC	XIOC-SER
xSysCom200.lib SysLibCom.lib	× ×	– –	– –	× ×
XC100_SysLibCom.lib	–	×	–	×
XN-PLC-SysLibCom.lib	–	–	×	–

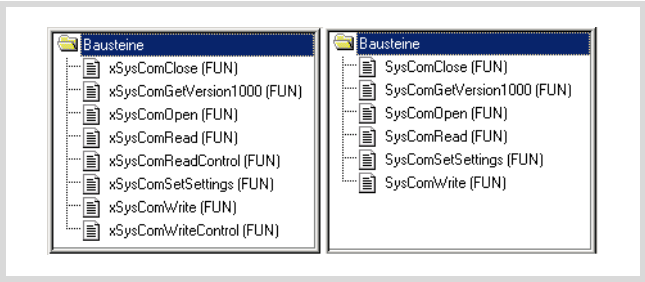


Figure 34: Function summary

General

With these functions you can configure the RS 232 port of the following devices and send and receive data from the user program.

- CPU: XC200,XC100,XN-PLC
- XIOC-SER module in connection with XC200/XC100

For XC200 only:
For compatibility reasons, two libraries are available. Only one of the two libraries can be integrated into the library manager. Both libraries contain functions, for example to open and close the interface. For a clear overview, the functions are shown next to each other in the following illustrations.

The left side shows the functions from the library for XC200, XC100 and XN-PLC; the right side shows the alternative functions from library "SysLibCom" for the XC200.

XC200	xSysCom200.lib	SysLibCom.lib
XC100	XC100_SysLibCom.lib	
XN-PLC	XN-PLC-SysLibCom.lib	

The function names in SysLibCom.lib are also used in xSysCom200.lib, but are prefixed with an "x", for example:

	xSysCom200.Lib	SysLibCom.lib
Function name	xSysComClose	SysComClose

If both functions are addressed, for example in the descriptions below, the function name is marked with an (x) for example (x)SysComClose.

Function "(x)SysComClose"

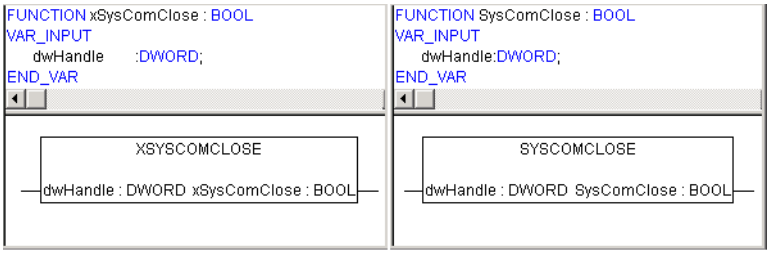


Figure 35: Function "(x)SysComClose

Description

The function closes the RS232 interface. The communication parameters which were set last are restored during closing. The function provides TRUE as a return value if the action is completed successfully.

Parameters

dwHandle	Return value of the "(x)SysComOpen" function
(x)SysComClose	Return value TRUE: closing the RS232 interface was successful

Function "(x)SysComOpen"

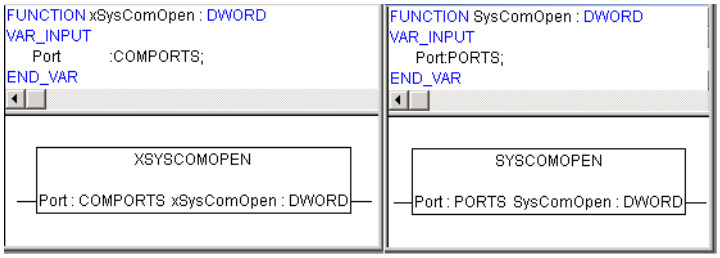


Figure 36: Function "(x)SysComOpen"

Description

The function opens the RS232 interface for transparent mode. After successful opening of the interface the function returns a value greater than "0".

- ▶ Enter this value for the following functions as the "dwHandle" parameter.

If a fault occurs, the return value is equal to "0". Transparent mode of the interface will not be enabled.

After opening of the RS232 interface the parameters can be set with the assistance of Function "(x)SysComSetSettings" (→ page 99). The values set beforehand for the XIOC-SER in the PLC configurator, and the default settings of the CPU are then ignored and are only valid again after the RS232 interface is closed.

Parameters

Port	Interface selection Parameters:	Specified for the open interface.
(x)SysComOpen	Return value 0:	Opening of the RS232 interface was **not** successful.
	Return value > 0:	Opening of the RS232 interface was successful.

Data types

XC200

```
TYPE COMPORTS :
(   COM1  :=1,
    COM2,
    COM3,
    COM4,
    COM5 ) := COM1;
END_TYPE
```

```
TYPE PORTS : (COM1:=1, COM2, COM3, COM4, COM5);
END_TYPE
```

XC100

```
TYPE COMPORTS :
(   COM1  :=1,
    COM2,
    COM3 ) := COM1;
END_TYPE
```

XN-PLC

```
TYPE COMPORTS :
 ( COM1    :=1 ) := COM1;
END_TYPE
```

Figure 37: COMPORTS/PORTS data types

Function "(x)SysComRead"

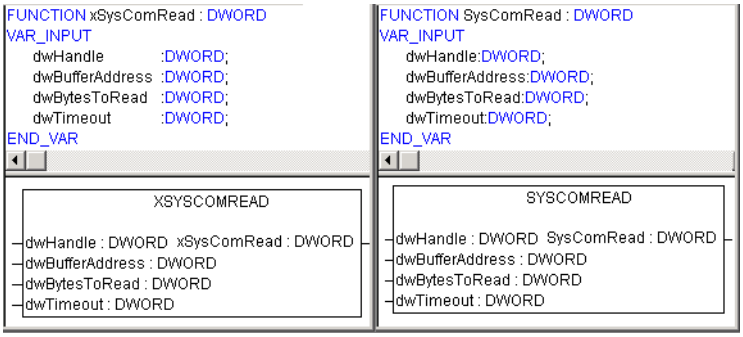


Figure 38: Function "(x)SysComRead"

Description

Data received via the RS232 interface in transparent mode can be read with this function.

Parameters

dwHandle	Return value of the "(x)SysComOpen" function
dwBufferAddress	Address under which the read data is stored
dwBytesToRead	Limitation of the max. number of data bytes (COM 2 to COM 5: max. 250 bytes)
dwTimeout	Parameters without meaning
(x)SysComRead	Return value provides information about the number of read data bytes.

Important
A test of the buffer address or buffer size does not occur!

Function "xSysComReadControl"

```
FUNCTION xSysComReadControl : BOOL
VAR_INPUT
    dwHandle    :DWORD;
    Control     :POINTER TO COMCONTROL;
END_VAR
```

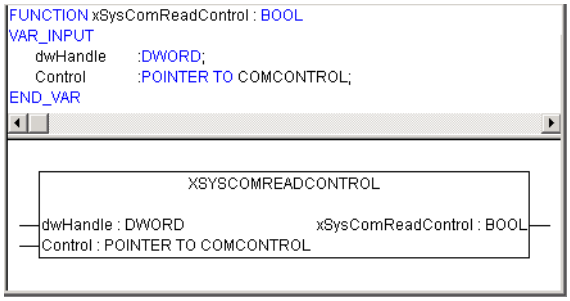


Figure 39: Function "xSysComReadControl"

Description

The XIOC-SER hardware interface module avails of control/ interface lines. It provides the "xSysComReadControl" module with read access to the control/interface lines of the COM 2 to COM 5 interfaces.

Parameters

dwHandle	Return value from the "xSysComOpen" function	
Control	COM 2 to COM 5:	TRUE = read command on the control lines of the hardware interface
xSysComReadControl	COM 2 to COM 5:	TRUE = read command was successful; FALSE = read command was unsuccessful

Data type

```
TYPE COMCONTROL :
STRUCT
    bRTS    :BOOL;
    bDTR    :BOOL;
    bDSR    :BOOL;
    bCTS    :BOOL;
    bRI     :BOOL;
    bDCD    :BOOL;
    bERR    :BOOL;
END_STRUCT
END_TYPE
```

Figure 40: COMCONTROL data type

Function "(x)SysComSetSettings"

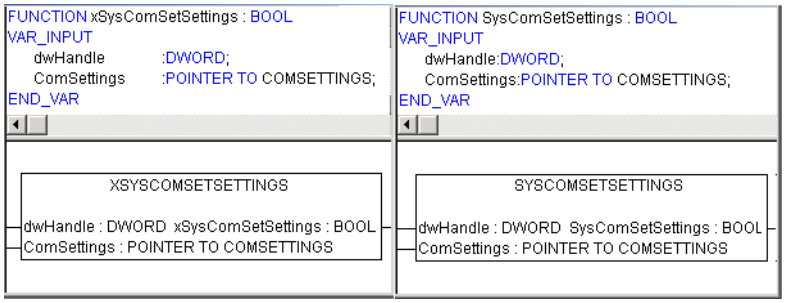


Figure 41: Function "(x)SysComSetSettings"

Description

Interface parameters of the RS232 interface for the transparent mode can be set with this function.

Parameters

dwHandle	Return value of the "(x)SysComOpen" function
ComSettings	Pointer to the memory area in which the interface parameters are stored
(x)SysComSetSettings	TRUE return value if the interface has been successfully parameterized; otherwise FALSE

Data types

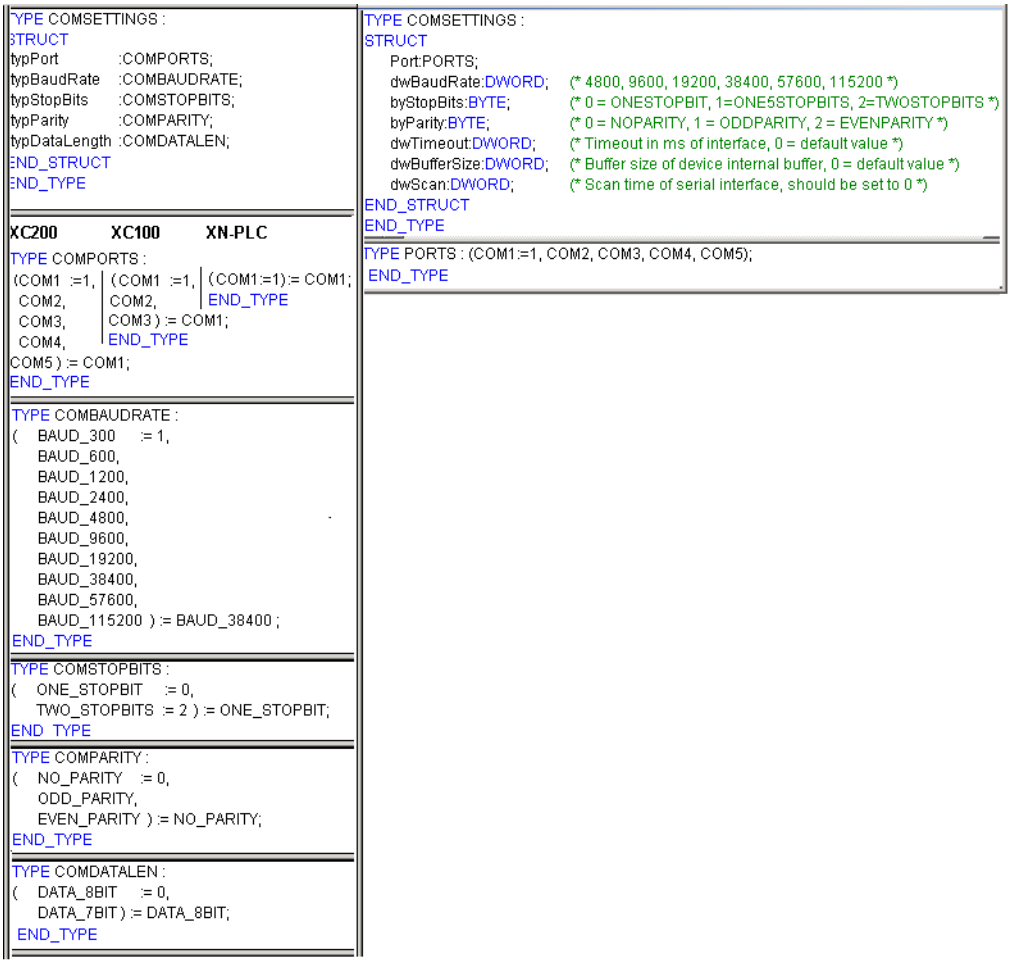


Figure 42: COMSETTINGS data type

Function "(x)SysComWrite"

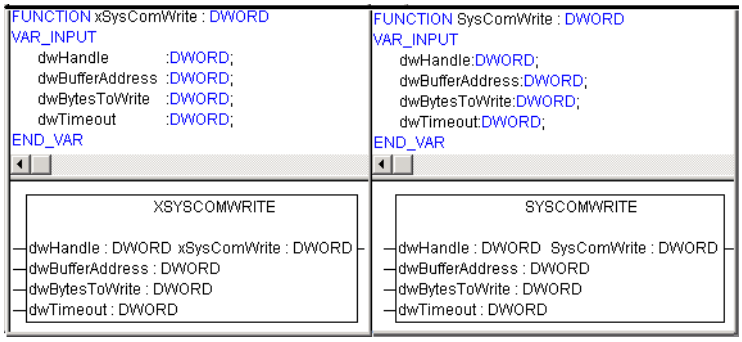


Figure 43: Function "(x)SysComWrite"

Description

This function allows output of data via the RS232 interface.

Parameters

dwHandle	Return value of the "(x)SysComOpen" function
dwBufferAddress	Address under which the data to be output is stored
dwBytesToWrite	Number of data bytes to be sent (COM 2 to COM 5: max. 250 bytes)
dwTimeout	Parameters without meaning
(x)SysComWrite	Return value provides information about the amount of sent data.

Important
A test of the buffer address or buffer size does not occur!

Function "(x)SysComWriteControl"

→ This function can only be used with the XIOC-SER module!

The XIOC-SER hardware interface module avails of control/ interface lines. It provides the "SysComWriteControl" module with write access to the control/interface lines of the COM 2 to COM 5 interfaces.

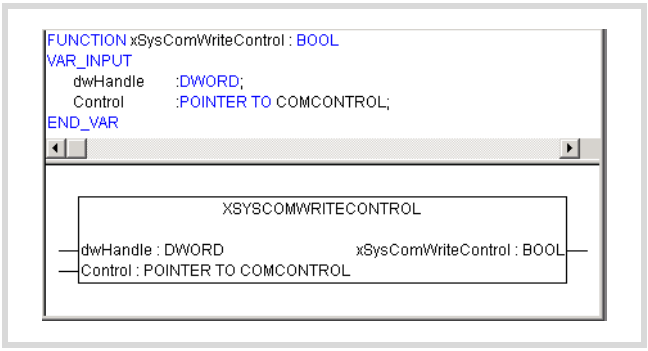


Figure 44: Write access to the control lines of the COM 2 to COM 5 interface

Parameters

dwHandle	Return value of the "(x)SysComOpen" function	
Control	COM 2 to COM 5:	TRUE = write command on the control lines of the hardware interface
(x)SysComReadControl	COM 2 to COM 5:	TRUE = write command was successful; FALSE = write command was unsuccessful

Data type

```
TYPE COMCONTROL :
STRUCT
    bRTS      :BOOL;
    bDTR      :BOOL;
    bDSR      :BOOL;
    bCTS      :BOOL;
    bRI       :BOOL;
    bDCD      :BOOL;
    bERR      :BOOL;
END_STRUCT
END_TYPE
```

Figure 45: COMCONTROL data type

Automatic closing of the interface

With a state change of the XC200 to STOP the transparent mode is ended automatically by the operating system. The interface is initialised again with the interface parameters last set.

Example
The example shows a text output via the RS232 interface of the XC-CPU201 in transparent mode.

```
PROGRAM PLC_PRG
VAR
     BRAKE:TONE;
     STEP:UINT;
     dwSioHandle: DWORD;
     WriteBuffer:STRING(26);
     nWriteLength: DWORD;
     typComSettings:COMSETTINGS;
     typComSetSettings:BOOL;
     out AT %QB0:BYTE;
     INP AT %IX0.0:BOOL;
     STEPERR: UINT;
     Closeresult: BOOL;
     Coun: DWORD;
     RESET: BOOL;
END_VAR

(*Cycle time/Cycletime: 50ms!*)
CASE STEP OF
0:   IF INP =1 THEN (*Start: IX0.0 = TRUE*)
     STEP:=1;
     END_IF
1:   (*Öffnen/Open*)
     IF dwSioHandle=0 THEN
         dwSioHandle:=xSysComOpen(Port:=Com1);
             IF (dwSioHandle>0) THEN
                   typComSettings.typBaudRate                          :=Baud_9600;
                   typComSettings.typDataLength                        :=Data_8Bit;
                   typComSettings.typParity                            :=NO_PARITY;
                   typComSettings.typPort                              :=COM1;
                   typComSettings.typStopBits                          :=ONE_STOPBIT;
                   xSysComSetSettings(dwHandle:=dwSioHandle,
                   ComSettings:=ADR(typComSettings));
                   STEP :=2;
                   RESET:=TRUE;
             ELSE
                   STEPERR:=STEP;
                   STEP :=99;
             END_IF
             WriteBuffer:='This is the sent text';
     END_IF
```

```
2: (*Ausgabe/Output*)
     IF (dwSioHandle>0) THEN
       nWriteLength:=xSysComWrite(dwHandle:=dwSioHandle,
       dwBufferAddress:=ADR(WriteBuffer),
       dwBytesToWrite:=LEN(WriteBuffer)+1,dwTimeOut:=0);
     END_IF
     IF nWriteLength = LEN(WriteBuffer)+1 THEN
        STEP :=3;
        Coun:=coun+1;
     END_IF
3: (*Schliessen/Close*)
     Closeresult:=xSysComClose(dwHandle:=dwSioHandle);
     IF (Closeresult = TRUE) THEN
        dwSioHandle:=0;
        STEP :=4;
     ELSE
        STEPERR:=STEP;
        STEP :=99;
     END_IF
4: (*Verzögerung/Delay*)
     BRAKE(IN:=1, PT:=T#2s);
     IF BRAKE.Q = 1 THEN
        STEP :=5;
        BRAKE(IN:=0, PT:=T#2s);
     END_IF
5: (*End*)
     STEP :=0;
99: (*Fehler/Error*)
     STEPERR:=STEPERR;
END_CASE
```

Index