

Concept  
IEC block library  
Part: EXTENDED

840 USE 504 00 eng Version 2.6

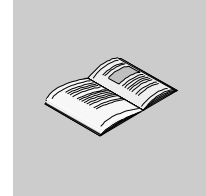


© 2002 Schneider Electric All Rights Reserved

---

---

## Table of Contents



---

<b>About the book</b> .....	<b>7</b>
<b>Part I General information on the block library EXTENDED</b> ..	<b>9</b>
Overview .....	9
<b>Chapter 1 Parameterizing functions and function blocks</b> .....	<b>11</b>
Parameterizing functions and function blocks .....	11
<b>Part II EFB-descriptions</b> .....	<b>15</b>
Overview .....	15
<b>Chapter 2 AVE_***: Averaging</b> .....	<b>19</b>
<b>Chapter 3 AVGMV: Floating mean with fixed window size</b> .....	<b>23</b>
<b>Chapter 4 AVGMV_K: Floating mean with frozen correction factor</b> ....	<b>27</b>
<b>Chapter 5 BCD_TO_INT: Conversion from 16 Bit BCD to INT</b> .....	<b>31</b>
<b>Chapter 6 BIT_TO_BYTE: Type conversion</b> .....	<b>33</b>
<b>Chapter 7 BIT_TO_WORD: Type conversion</b> .....	<b>37</b>
<b>Chapter 8 BYTE_AS_WORD: Type conversion</b> .....	<b>41</b>
<b>Chapter 9 BYTE_TO_BIT: Type conversion</b> .....	<b>43</b>
<b>Chapter 10 CTD_***: Down counter</b> .....	<b>45</b>
<b>Chapter 11 CTU_***: Up counter</b> .....	<b>47</b>
<b>Chapter 12 CTUD_***: Up/Down counter</b> .....	<b>49</b>
<b>Chapter 13 DBCD_TO_DINT: Conversion from 32 Bit BCD to DINT</b> .....	<b>53</b>
<b>Chapter 14 DBCD_TO_INT: Conversion from 32 Bit BCD to INT</b> .....	<b>55</b>
<b>Chapter 15 DEAD_ZONE_REAL: Dead zone</b> .....	<b>57</b>

---

Chapter 16	DINT_AS_WORD: Type conversion . . . . .	61
Chapter 17	DINT_TO_DBCD: Conversion from DINT to 32 Bit BCD . . . . .	63
Chapter 18	DIVMOD_***: Division and Modulo . . . . .	65
Chapter 19	HYST_***: Indicator signal for maximum value delimiter with hysteresis . . . . .	67
Chapter 20	INDLIM_***: Indicator signal for delimiters with hysteresis . .	71
Chapter 21	INT_TO_BCD: Conversion from INT to 16 Bit BCD . . . . .	75
Chapter 22	INT_TO_DBCD: Conversion from INT to 32 Bit BCD . . . . .	77
Chapter 23	LIMIT_IND_***: Limit with indicator . . . . .	79
Chapter 24	LOOKUP_TABLE1: Traverse progression with 1st degree interpolation . . . . .	83
Chapter 25	NEG_***: Negation . . . . .	87
Chapter 26	REAL_AS_WORD: Type conversion . . . . .	91
Chapter 27	SAH: Detecting and holding with rising edge . . . . .	93
Chapter 28	SIGN_***: Sign evaluation . . . . .	95
Chapter 29	TIME_AS_WORD: Type conversion . . . . .	99
Chapter 30	TOF_P: Off Delay with Pause . . . . .	101
Chapter 31	TON_P: On Delay with Pause . . . . .	105
Chapter 32	TRIGGER: Detection of all types of edges . . . . .	109
Chapter 33	UDINT_AS_WORD: Type conversion . . . . .	111
Chapter 34	WORD_AS_BYTE: Type conversion . . . . .	113
Chapter 35	WORD_AS_DINT: Type conversion . . . . .	115
Chapter 36	WORD_AS_REAL: Type conversion . . . . .	117
Chapter 37	WORD_AS_TIME: Type conversion . . . . .	119
Chapter 38	WORD_AS_UDINT: Type conversion . . . . .	121
Chapter 39	WORD_TO_BIT: Type conversion . . . . .	123
	Glossary . . . . .	127

---

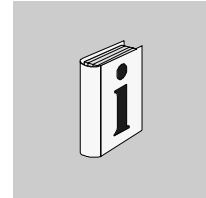
---

<b>Index</b>	.....	<b>151</b>
--------------	-------	------------

---

---

## About the book



---

### At a Glance

**Document Scope** This documentation is designed to help with the configuration of functions and function blocks.

**Validity Note** This documentation applies to Concept 2.6 under Microsoft Windows 98, Microsoft Windows 2000 and Microsoft Windows NT 4.x.

**Note:** There is additional up to date tips in the README data file in Concept.

---

### Related Documents

Title of Documentation	Reference Number
Concept Installation Instructions	840 USE 502 00
Concept User Manual	840 USE 503 00
Concept EFB User Manual	840 USE 505 00
Concept LL984 Block Library	840 USE 506 00

**User Comments** We welcome your comments about this document. You can reach us by e-mail at [TECHCOMM@modicon.com](mailto:TECHCOMM@modicon.com)

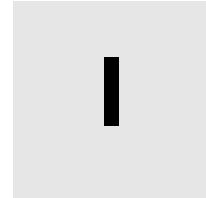
---

About the book

---

---

## General information on the block library EXTENDED



---

### Overview

#### Introduction

This section contains general information on the block library EXTENDED.

#### What's in this part?

This part contains the following chapters:

Chapter	Chaptername	Page
1	Parameterizing functions and function blocks	11

## General information

---

---

## **Parameterizing functions and function blocks**



**1**

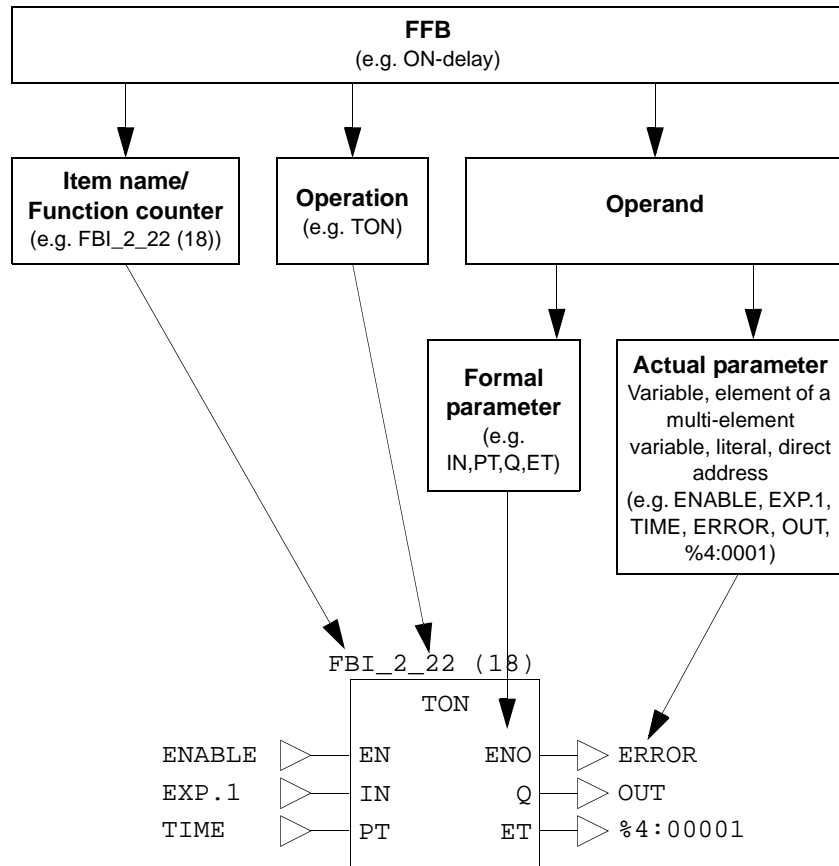
---

**Parameterizing functions and function blocks**

---

**General**

Each FFB consists of an operation, the operands needed for the operation and an instance name or function counter.



**Operation**

The operation determines which function is to be executed with the FFB, e.g. shift register, conversion operations.

**Operand**

The operand specifies what the operation is to be executed with. With FFBs, this consists of formal and actual parameters.

---

**Formal/actual parameters**

The formal parameter holds the place for an operand. During parameterization, an actual parameter is assigned to the formal parameter.

The actual parameter can be a variable, a multi-element variable, an element of a multi-element variable, a literal or a direct address.

---

**Conditional/unconditional calls**

"Unconditional" or "conditional" calls are possible with each FFB. The condition is realized by pre-linking the input EN.

- Displayed EN  
conditional calls (the FFB is only processed if EN = 1)
- EN not displayed  
unconditional calls (FFB is always processed)

**Note:** If the EN input is not parameterized, it must be disabled. Any input pin that is not parameterized is automatically assigned a "0" value. Therefore, the FFB should never be processed.

---

**Calling functions and function blocks in IL and ST**

Information on calling functions and function blocks in IL (Instruction List) and ST (Structured Text) can be found in the relevant chapters of the user manual.

---



---

## EFB-Descriptions



---

### Overview

#### Introduction

These EFB descriptions are documented in alphabetical order.

**Note:** The number of inputs of the EFBs can be increased to a maximum of 32 by changing the size of the FFB symbols vertically. Please consult the individual EFB descriptions to know which EFBs are concerned.

**What's in this part?**

This part contains the following chapters:

Chapter	Chaptername	Page
2	AVE_***: Averaging	19
3	AVGMV: Floating mean with fixed window size	23
4	AVGMV_K: Floating mean with frozen correction factor	27
5	BCD_TO_INT: Conversion from 16 Bit BCD to INT	31
6	BIT_TO_BYTE: Type conversion	33
7	BIT_TO_WORD: Type conversion	37
8	BYTE_AS_WORD: Type conversion	41
9	BYTE_TO_BIT: Type conversion	43
10	CTD_***: Down counter	45
11	CTU_***: Up counter	47
12	CTUD_***: Up/Down counter	49
13	DBCD_TO_DINT: Conversion from 32 Bit BCD to DINT	53
14	DBCD_TO_INT: Conversion from 32 Bit BCD to INT	55
15	DEAD_ZONE_REAL: Dead zone	57
16	DINT_AS_WORD: Type conversion	61
17	DINT_TO_DBCD: Conversion from DINT to 32 Bit BCD	63
18	DIVMOD_***: Division and Modulo	65
19	HYST_***: Indicator signal for maximum value delimiter with hysteresis	67
20	INDLIM_***: Indicator signal for delimiters with hysteresis	71
21	INT_TO_BCD: Conversion from INT to 16 Bit BCD	75
22	INT_TO_DBCD: Conversion from INT to 32 Bit BCD	77
23	LIMIT_IND_***: Limit with indicator	79
24	LOOKUP_TABLE1: Traverse progression with 1st degree interpolation	83
25	NEG_***: Negation	87
26	REAL_AS_WORD: Type conversion	91
27	SAH: Detecting and holding with rising edge	93
28	SIGN_***: Sign evaluation	95
29	TIME_AS_WORD: Type conversion	99
30	TOF_P: Off Delay with Pause	101
31	TON_P: On Delay with Pause	105
32	TRIGGER: Detection of all types of edges	109
33	UDINT_AS_WORD: Type conversion	111

<b>Chapter</b>	<b>Chaptername</b>	<b>Page</b>
34	WORD_AS_BYTE: Type conversion	113
35	WORD_AS_DINT: Type conversion	115
36	WORD_AS_REAL: Type conversion	117
37	WORD_AS_TIME: Type conversion	119
38	WORD_AS_UDINT: Type conversion	121
39	WORD_TO_BIT: Type conversion	123

---



---

## AVE\_\*\*\*: Averaging



2

---

### Overview

#### Introduction

This chapter describes the AVE\_\*\*\* block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	20
Representation	21

---

## Brief description

---

### Function description

The Function calculates the mean of weighted input values, and the result is then given at the output.

Each two successive inputs (K\_Xn) represent one pair of values. The first K\_Xn input corresponds to K1, the next to X1, the one after that to K2, etc.

The number of K\_Xn inputs can be increased to 32 by vertically modifying the size of the block frame. This corresponds to a maximum of 16 value pairs.

The number of inputs must be even.

Data types of the ANY\_NUM group can be processed.

The data types of all input and output values must be identical. A specific function is available to process each of the different data types.

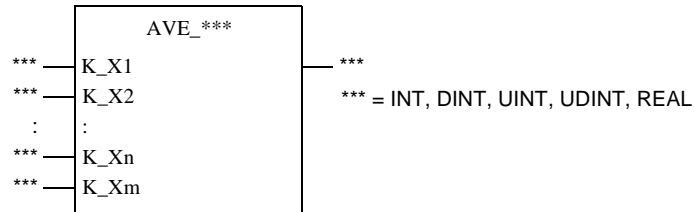
It is possible to project EN and ENO as additional parameters.

---

## Representation

### Symbol

Block representation:



### Formula

Block formula:

$$OUT = \frac{\sum(K_i \times X_i)}{\sum(K_i)}$$

### Parameter description

Block parameter description:

Parameters	Data type	Meaning
K_X1	INT, DINT, UINT, UDINT, REAL	Factor (K1) for first value
K_X2	INT, DINT, UINT, UDINT, REAL	First value (X1)
K_X3	INT, DINT, UINT, UDINT, REAL	Factor (K2) for second value
K_X4	INT, DINT, UINT, UDINT, REAL	Second value (X2)
:		
K_Xn	INT, DINT, UINT, UDINT, REAL	Factor ( $K_{m/2}$ ) for m/2 value
K_Xm	INT, DINT, UINT, UDINT, REAL	m/2 value ( $X_{m/2}$ )
OUT	INT, DINT, UINT, UDINT, REAL	Mean value

AVE\_\*\*\*: Averaging

---

---

## AVGMV: Floating mean with fixed window size

3

---

### Overview

#### Introduction

This chapter describes the AVGMV block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	24
Representation	24
Detailed description	25
Runtime error	26

## Brief description

---

### Function description

The function block forms a floating mean from a fixed number of input values (Input X). The output is the mean of all values between the current X value and the oldest X-value (N-1). It is possible to save up to 50 input values (N). The function block can operate in manual and automatic mode. It is possible to project EN and ENO as additional parameters.

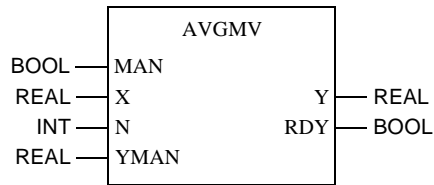
---

## Representation

---

### Symbol

Block representation:



### Formula

With RDY = 1:

$$Y_{(new)} = \frac{\sum_{i=0}^{N-1} X_i}{N}$$

or

$$Y_{(new)} = Y_{(old)} + \frac{X}{N} - \frac{X_{(N-1)}}{N}$$

Explanation of variables

Variable	Meaning
$Y_{(new)}$	Y value in current program cycle
$Y_{(old)}$	Y value from last program cycle
N	Window size (number of values in buffer)
$X_{(N-1)}$	oldest X value in buffer

---

**Parameter description**

Block parameter description:

Parameter	Data type	Meaning
MAN	BOOL	"0" = automatic operating mode "1" = manual operating mode
X	REAL	Input
N	INT	Window size (number of input values that are loaded into the buffer; 50 max.)
YMAN	REAL	Manual value
Y	REAL	Mean value
RDY	BOOL	"1" = n values in buffer, i.e. buffer ready "0" = buffer not ready

**Detailed description**

**Automatic operating mode**

In N program cycles, N X-values are read into an internal buffer. The arithmetic mean of these values is calculated, and is delivered at the Y output. From the N+1 program cycle onwards, the oldest X-value in the buffer is deleted and replaced with the current x-value.

**Note:** As long as RDY = 0, the mean is not derived from N values but from the current updated read-in number (n < N).

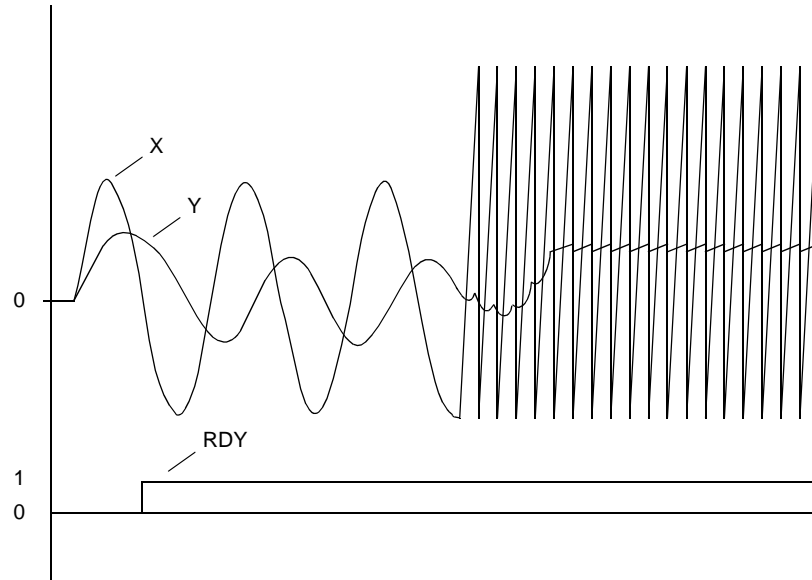
After a modification of the N value or after a cold/warm start, the internal buffer is deleted. The output is set to the input value X and RDY to "0". The buffer is filled during the next N cycles. The Y output contains an mean of the values accumulated so far. RDY remains "0" until the buffer is filled with correct X values after N program cycles then RDY becomes "1".

**Manual operating mode**

The value YMAN is transferred to the Y output. The buffer is completely filled with the value YMAN and marked as full (RDY = 1).

**Diagram**

Floating mean with limited memory of  $N = 50$  values



---

**Runtime error**

**Runtime error**

An Error message appears if,

- $N = 0$  or  $N > 50$

---

---

## AVGMV\_K: Floating mean with frozen correction factor

# 4

---

### Overview

#### Introduction

This chapter describes the AVGMV\_K block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	28
Representation	28
Detailed description	29

---

## Brief description

---

### Function description

The function block establishes a floating mean (with frozen correction factor) of up to 10 000 input values.  
 The function block can operate in manual and automatic mode.  
 It is possible to project EN and ENO as additional parameters.

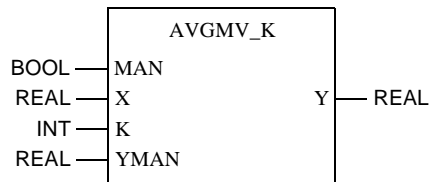
---

## Representation

---

### Symbol

Block representation:



### Formula

Block formula:

$$Y_{(new)} = Y_{(old)} + \frac{X - Y_{(old)}}{K}$$

Explanation of variables

Variable	Meaning
$Y_{(new)}$	Y value in current program cycle
$Y_{(old)}$	Y value from last program cycle
K	Correction factor
X	X value in current program cycle

---

### Parameter description

Block parameter description:

Parameter	Data type	Meaning
MAN	BOOL	"0" = Automatic Mode; "1" = Manual Mode
X	REAL	Input
K	INT	Correction factor (max. 10 000)
YMAN	REAL	Manual value
Y	REAL	Mean value

## Detailed description

### Automatic operating mode

One X value is read in every program cycle.  $1/N$  is deducted from the Y value of the last program cycle, and then  $1/N$  of the current X value is added. The result is delivered at the Y output.

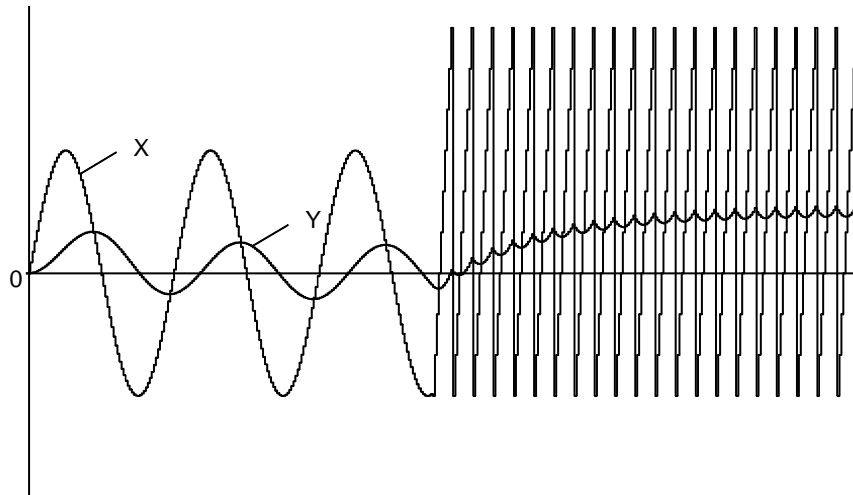
After a cold or warm (re)start, the X value is assigned to output Y.

### Manual operating mode

The value YMAN is transferred to the Y output.

### Diagram

Floating mean with frozen correction factor ( $K = 50$ )



AVGMV\_K: Floating mean with frozen correction factor

---

---

## BCD\_TO\_INT: Conversion from 16 Bit BCD to INT



---

### Overview

#### Introduction

This chapter describes the BCD\_TO\_INT block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	32
Representation	32

---

## Brief description

---

### Function description

This function converts a 16 Bit BCD input value (8-4-2-1-Code) into an INT data type output value.  
 If no valid BCD coded value is created at input, a runtime error is reported and the input value passes unchanged to the output.  
 EN and ENO can be configured as additional parameters.

---

### Example

Example of a BCD -> INT conversion

	Data type	Dec. value	Hex. value (= BCD-value)
<b>Input value</b>	INT	-26 797	9753
<b>Output value</b>	INT	9 753	-

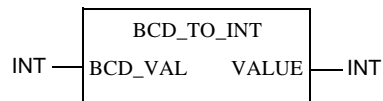
---

## Representation

---

### Symbol

Block representation:



### Parameter description

Block parameter description:

Parameter	Data type	Meaning
BCD_VAL	INT	16 Bit BCD input value
VALUE	INT	INT output value

---

---

## BIT\_TO\_BYTE: Type conversion



---

### Overview

#### Introduction

This chapter describes the BIT\_TO\_BYTE block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	34
Representation	35

---

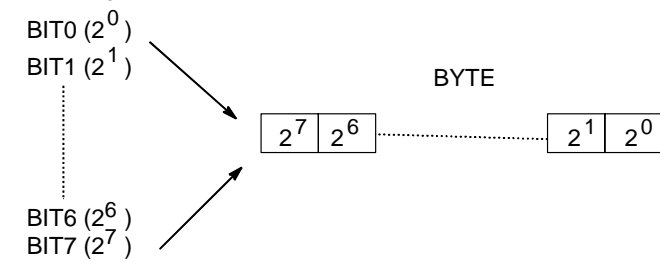
## Brief description

---

### Function description

The Function converts 8 input values of the Data type BOOL to an output of the data type BYTE.

The input values are assigned to the individual bits of the byte at the output according to the input names.



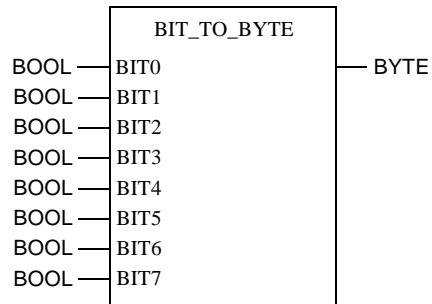
It is possible to project EN and ENO as additional parameters.

---

## Representation

### Symbol

Block representation:



### Formula

Block formula:

$$OUT = \{BIT7, BIT6, \dots, BIT0\}$$

### Parameter description

Block parameter description:

Parameter	Data type	Meaning
BIT0	BOOL	Input bit 0
BIT1	BOOL	Input bit 1
:	:	:
BIT7	BOOL	Input bit 7
OUT	BYTE	Output value

BIT\_TO\_BYTE: Type conversion

---

---

## BIT\_TO\_WORD: Type conversion



---

### Overview

#### Introduction

This chapter describes the BIT\_TO\_WORD block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	38
Representation	39

---

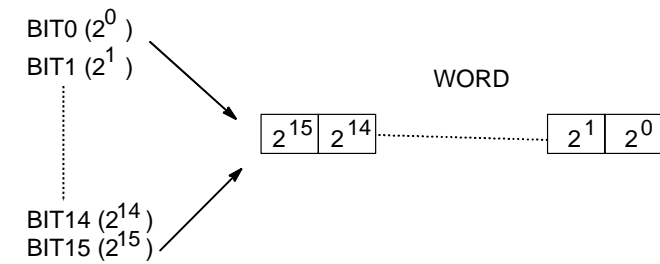
## Brief description

---

### Function description

The Function converts 16 input words from Data type BOOL to an output value of data type WORD.

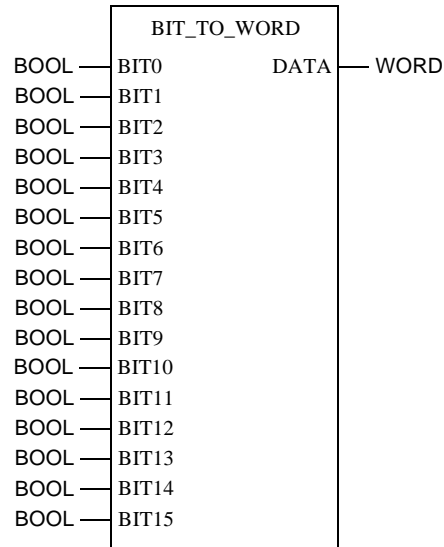
The input values are assigned to the individual bits of the word at the output according to the input names.



## Representation

### Symbol

Block representation:



### Formula

Block formula:

$$OUT = \{BIT15, BIT14, \dots, BIT0\}$$

### Parameter description

Block parameter description:

Parameter	Data type	Meaning
BIT0	BOOL	Input bit 0
BIT1	BOOL	Input bit 1
:	:	:
BIT15	BOOL	Input bit 15
OUT	WORD	Output value

BIT\_TO\_WORD: Type conversion

---

---

## BYTE\_AS\_WORD: Type conversion



---

### Overview

#### Introduction

This chapter describes the BYTE\_AS\_WORD block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	42
Representation	42

---

## Brief description

---

### Function description

The Function converts 2 input words from Data type BYTE to an output value of data type WORD.  
 The input values are assigned to the word at the output according to the input names.  
 It is possible to project EN and ENO as additional parameters.

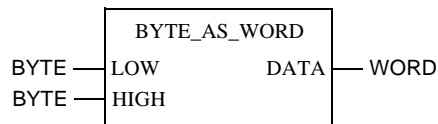
---

## Representation

---

### Symbol

Block representation:



### Formula

Block formula:

$$OUT = \{HIGH, LOW\}$$


---

### Parameter description

Block parameter description:

Parameter	Data type	Meaning
LOW	BYTE	less significant byte
HIGH	BYTE	more significant byte
OUT	WORD	Output value

---

---

## BYTE\_TO\_BIT: Type conversion

9

---

### Overview

#### Introduction

This chapter describes the BYTE\_TO\_BIT block.

#### What's in this chapter?

This chapter contains the following topics:

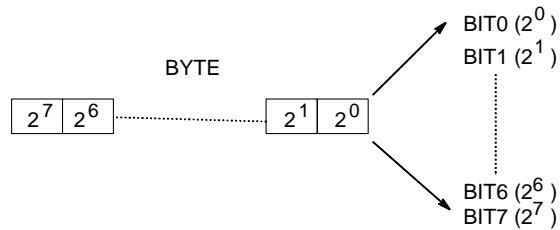
Topic	Page
Brief description	44
Representation	44

---

## Brief description

### Function description

This function block converts one input word from Data type BYTE into 8 output values of data type BOOL. The individual bits of the byte at the input are assigned to the outputs according to the output names.

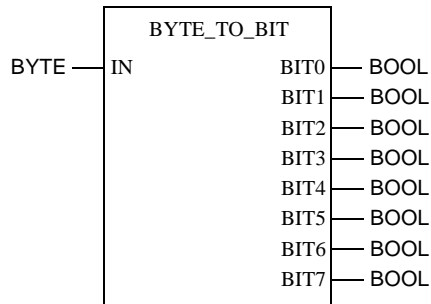


It is possible to project EN and ENO as additional parameters.

## Representation

### Symbol

Block representation:



### Parameter description

Block parameter description:

Parameter	Data type	Meaning
IN	BYTE	Input
BIT0	BOOL	Output bit 0
BIT1	BOOL	Output bit 1
:	:	:
BIT7	BOOL	Output bit 7

---

## CTD\_\*\*\*: Down counter

10

---

### Overview

#### Introduction

This chapter describes the CTD\_\*\*\* block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	46
Representation	46

## Brief description

### Function description

The Function block is used for counting down.  
 A "1" signal at the LD input causes the value of the PV input to be allocated to the CV output. With each transition from "0" to "1" at the CD input, the value of CV is reduced by 1.  
 When CV is  $\leq 0$ , the output Q becomes "1".  
 The Data types of the PV input and the CV output must be identical. A specific function block is available to process each of the different data types.

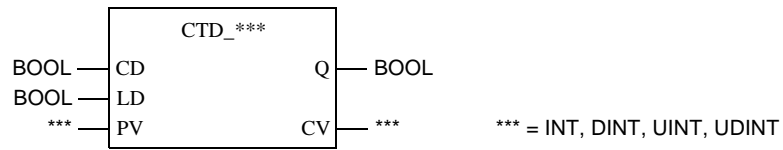
**Note:** The counter only works up to the minimum values of the data type being used. No overflow occurs.

The parameters EN and ENO can additionally be projected.

## Representation

### Symbol

Block representation:



### Parameter description

Block parameter description:

Parameter	Data type	Meaning
CD	BOOL	Trigger input
LD	BOOL	Load data
PV	INT, DINT, UINT, UDINT	Presettings value
Q	BOOL	Output
CV	INT, DINT, UINT, UDINT	Count value (actual value)

---

## CTU\_\*\*\*: Up counter

11

---

### Overview

#### Introduction

This chapter describes the CTU\_\*\*\* block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	48
Representation	48

## Brief description

---

### Function description

The Function block is used for counting up.  
 A "1" signal at the R input causes the value "0" to be allocated to the CV output. With each transition from "0" to "1" at the CU input, the value of CV is increased by 1. When  $CV \geq PV$ , the Q output is set to "1".  
 The Data types of the PV input and the CV output must be identical. A specific function block is available to process each of the different data types.

**Note:** The counter only works up to the maximum values of the data type being used. No overflow occurs.

The parameters EN and ENO can additionally be projected.

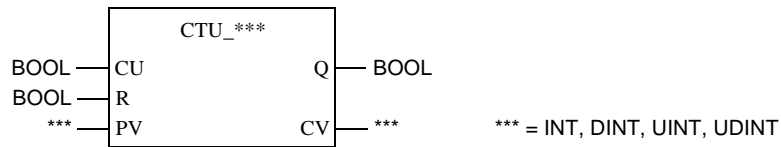
---

## Representation

---

### Symbol

Block representation:



### Parameter description

Block parameter description:

Parameter	Data type	Meaning
CU	BOOL	Trigger input
R	BOOL	Reset
PV	INT, DINT, UINT, UDINT	Presettings value
Q	BOOL	Output
CV	INT, DINT, UINT, UDINT	Count value (actual value)

---

---

## CTUD\_\*\*\*: Up/Down counter

12

---

### Overview

#### Introduction

This chapter describes the CTUD\_\*\*\* block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	50
Representation	51

## Brief description

---

### Function description

The Function block is used for counting up and down.

A "1" signal at the R input causes the value "0" to be allocated to the CV output. A "1" signal at the LD input causes the value of the PV input to be allocated to the CV output. With each transition from "0" to "1" at the CU input, the value of CV is increased by 1. With each transition from "0" to "1" at the CD input, the value of CV is reduced by 1.

If there is a simultaneous "1" signal at input R and input LD, input R has precedence.

When  $CV \geq PV$ , output QU is "1".

When  $CV \leq 0$ , output QD is "1".

**Note:** The down counter only works up to the minimum values of the data type being used, and the up counter only up to the maximum values of the data type being used. No overflow occurs.

The Data types of input PV and input CV must be identical. A specific Function block is available to process each of the different data types.

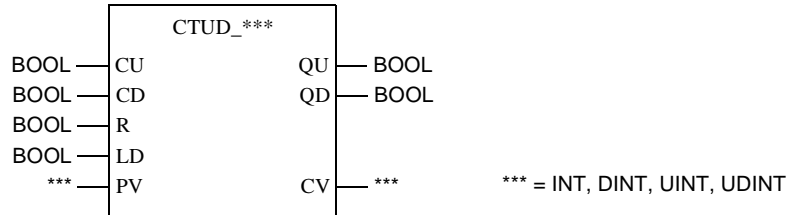
The parameters EN and ENO can additionally be projected.

---

## Representation

### Symbol

Block representation:



### Parameter description

Block parameter description:

Parameter	Data type	Meaning
CU	BOOL	Up counter trigger input
CD	BOOL	Down counter trigger input
R	BOOL	Reset
LD	BOOL	Load data
PV	INT, DINT, UINT, UDINT	Presettings value
QU	BOOL	Up display
QD	BOOL	Down display
CV	INT, DINT, UINT, UDINT	Count value (actual value)

CTUD\_\*\*\*: Up/Down counter

---

---

## DBCD\_TO\_DINT: Conversion from 32 Bit BCD to DINT

13

---

### Overview

#### Introduction

This chapter describes the DBCD\_TO\_DINT block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	54
Representation	54

---

## Brief description

---

### Function description

This function converts a 32 Bit BCD input value (8-4-2-1-Code) into a DINT data type output value.  
 If no valid BCD coded value is created at input, a runtime error is reported and the input value passes unchanged to the output.  
 EN and ENO can be configured as additional parameters.

---

### Example

Example of a DBCD -> DINT conversion

	Data type	Dec. value	Hex. value (= BCD-value)
<b>Input value</b>	DINT	-2 023 406 815	8765 4321
<b>Output value</b>	DINT	87 654 321	-

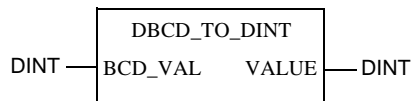
---

## Representation

---

### Symbol

Block representation:



### Parameter description

Block parameter description:

Parameter	Data type	Meaning
BCD_VAL	DINT	32 Bit BCD input value
VALUE	DINT	DINT output value

---

---

## DBCD\_TO\_INT: Conversion from 32 Bit BCD to INT

14

---

### Overview

#### Introduction

This chapter describes the DBCD\_TO\_INT block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	56
Representation	56

## Brief description

---

### Function description

This function converts a 32 Bit BCD input value (8-4-2-1-Code) into an INT data type output value.

The valid input value range is 0 32 767 (BCD.).

The following runtime errors are generated:

- If no valid BCD coded value is created at input, a runtime error is reported and the input value passes unchanged to the output.
- If the BCD format is correct, but too large (> 32 767) a runtime error is reported and the output value is set to -1.

EN and ENO can be configured as additional parameters.

---

### Example

Example of a DBCD -> INT conversion

	Data type	Dec. value	Hex. value (= BCD-value)
<b>Input value</b>	DINT	22 083	0000 5643
<b>Output value</b>	INT	5 643	-

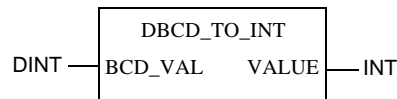
---

## Representation

---

### Symbol

Block representation:



### Parameter description

Block parameter description:

Parameter	Data type	Meaning
BCD_VAL	DINT	32 Bit BCD input value (valid value range 0 - 32767)
VALUE	INT	INT output value

---

---

## DEAD\_ZONE\_REAL: Dead zone

15

---

### Overview

#### Introduction

This chapter describes the DEAD\_ZONE\_REAL block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	58
Representation	58
Detailed description	59

DEAD\_ZONE\_REAL: Dead zone

---

## Brief description

---

### Function description

The Function is used to specify a deadzone for control variables. It is possible to project EN and ENO as additional parameters.

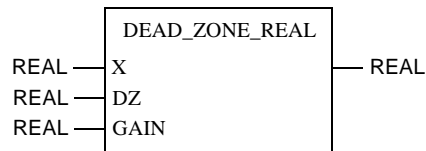
---

## Representation

---

### Symbol

Block representation:



### Formula

Block formula:  
Assuming:  $DZ \geq 0$   
 $Y = GAIN \times X$  for  $-DZ \leq X \leq DZ$   
 $Y = (X - DZ) + GAIN \times DZ$  for  $X > DZ$   
 $Y = (X + DZ) - GAIN \times DZ$  for  $X < -DZ$

---

### Parameter description

Block parameter description:

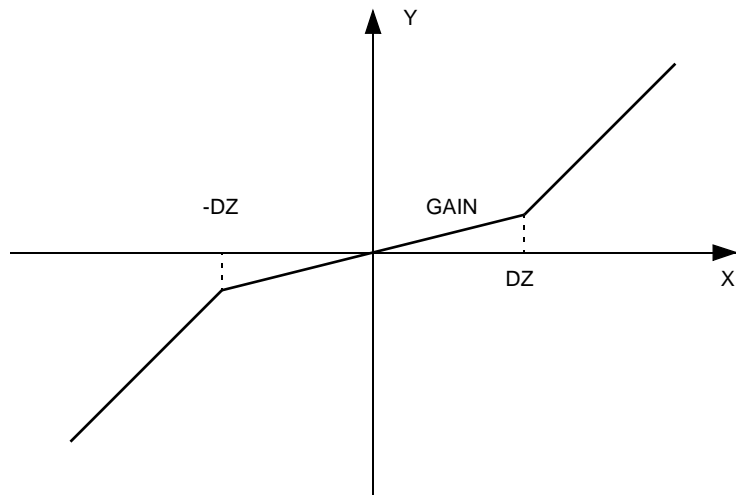
Parameter	Data type	Meaning
X	REAL	Input variable
DZ	REAL	Half width of the deadzone
GAIN	REAL	Gradient within deadzone
Y	REAL	Output variable

---

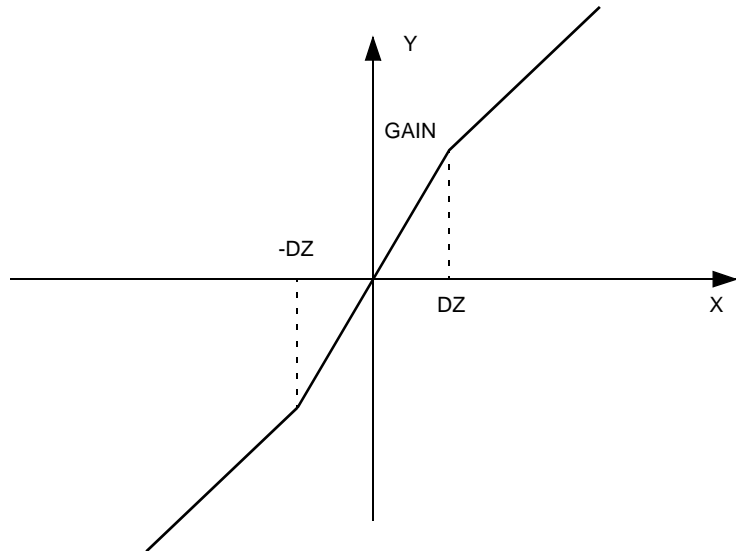
## Detailed description

### Characteristic Curves

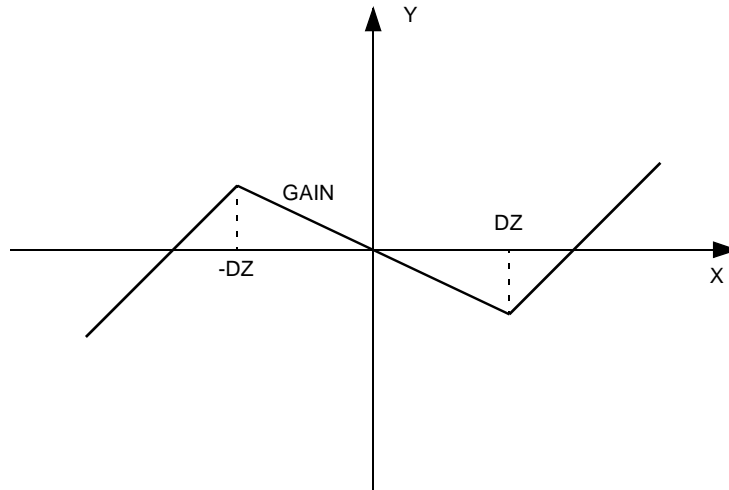
The function block has the following characteristic curve:  
Deadzone with  $0 < \text{GAIN} < 1$



Deadzone with  $\text{GAIN} > 1$



Deadzone with GAIN < 0



**Note:** Outside the dead zone, a gradient of 1 has been specified.

---

---

## DINT\_AS\_WORD: Type conversion

16

---

### Overview

#### Introduction

This chapter describes the DINT\_AS\_WORD block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	62
Representation	62

---

## Brief description

---

### Function description

This function block converts one input value of Data type DINT to 2 output values of data type WORD.  
The individual words of the DINT input are assigned to the outputs corresponding to the output names.  
It is possible to project EN and ENO as additional parameters.

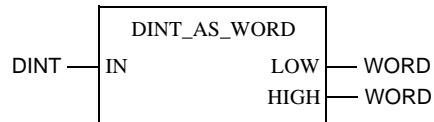
---

## Representation

---

### Symbol

Block representation:



### Parameter description

Block parameter description:

Parameter	Data type	Meaning
IN	DINT	Input
LOW	WORD	less significant word
HIGH	WORD	more significant word

---

---

## DINT\_TO\_DBCD: Conversion from DINT to 32 Bit BCD

17

---

### Overview

#### Introduction

This chapter describes the DINT\_TO\_DBCD block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	64
Representation	64

---

## Brief description

---

### Function description

This function converts a DINT data type input value into a 32 Bit BCD output value (8-4-2-1-Code).

The valid input value range is 0 99 999 999 (Dec.).

If no valid value is created at input, a runtime error is reported and the input value passes unchanged to the output.

EN and ENO can be configured as additional parameters.

---

### Example

Example of a DINT -> DBCD conversion

	Data type	Dec. value	Hex. value (= BCD-value)
<b>Input value</b>	DINT	87 654 321	-
<b>Output value</b>	DINT	-2 023 406 815	8765 4321

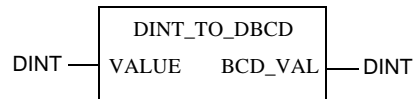
---

## Representation

---

### Symbol

Block representation:



### Parameter description

Block parameter description:

Parameter	Data type	Meaning
VALUE	DINT	DINT input value (valid value range 0 - 99 999 999)
BCD_VAL	DINT	32 Bit BCD output value

---

---

## DIVMOD\_\*\*\*: Division and Modulo

18

---

### Overview

#### Introduction

This chapter describes the DIVMOD\_\*\*\* block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	66
Representation	66
Runtime error	66

## Brief description

---

### Function description

This function block divides the value at input IN1 by the value at input IN2. The result of the division (quotient) is delivered at output DV. The remainder of the division (Modulo) is delivered at output MD.

If there is a decimal place in the division result, the division will truncate it.

Data types of the ANY\_INT group can be processed.

The data types of all input and output values must be identical. A specific function is available to process each of the different data types.

It is possible to project EN and ENO as additional parameters.

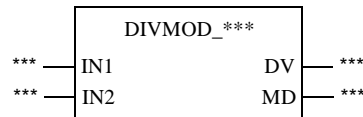
---

## Representation

---

### Symbol

Block representation:



\*\*\* = INT, DINT, UINT, UDINT

---

### Formula

Block formula:

$$DV = IN1 / IN2$$

$$MD = IN1 \text{ mod } IN2$$


---

### Parameter description

Block parameter description:

Parameter	Data type	Meaning
IN1	INT, DINT, UINT, UDINT	Dividend
IN2	INT, DINT, UINT, UDINT	Divisor
DV	INT, DINT, UINT, UDINT	Quotient
MD	INT, DINT, UINT, UDINT	Modulo

---

## Runtime error

---

### Runtime error

An Error message appears, when

- IN2=0
-

---

## **HYST\_\*\*\*: Indicator signal for maximum value delimiter with hysteresis**

**19**

---

### **Overview**

#### **Introduction**

This chapter describes the HYST\_\*\*\* block.

#### **What's in this chapter?**

This chapter contains the following topics:

<b>Topic</b>	<b>Page</b>
Brief description	68
Representation	68
Detailed Description	69

HYST\_\*\*\*: Indicator for maximum value delimiter with hysteresis

---

## Brief description

---

### Function description

The function block monitors the input variable X for violation of the upper threshold.

**Note:** If the lower threshold should be monitored as well, use the INDLIM function block.

Data types of the ANY\_NUM group can be processed.  
The data types of all input values must be identical. A specific function is available process each of the different data types.  
It is possible to project EN and ENO as additional parameters.

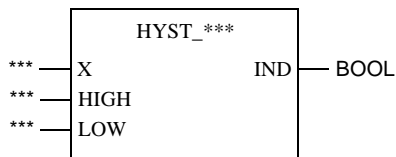
---

## Representation

---

### Symbol

Block representation:



\*\*\* = INT, DINT, UINT, UDINT, REAL

---

### Parameter description

Block parameter description:

Parameter	Data type	Meaning
X	INT, DINT, UINT, UDINT, REAL	Input variable
HIGH	INT, DINT, UINT, UDINT, REAL	Maximum upper threshold
LOW	INT, DINT, UINT, UDINT, REAL	Minimum upper threshold
IND	BOOL	Anzeige: reached upper threshold

---

## Detailed Description

---

### Parameter description

If X violates the upper HIGH limit, the function block reports this condition with IND = 1.  
If, subsequently, X violates the LOW threshold, IND will be reset to "0".

---

### Function

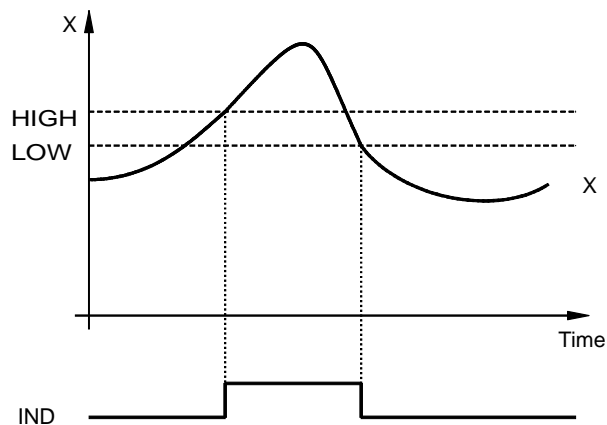
Function description  
 $IND_i = 1$ , if  $X > HIGH$   
 $IND_i = 0$ , if  $X < LOW$   
otherwise  
 $IND_i = IND_{i-1}$

If LOW is greater than HIGH, there is no hysteresis, and IND shows X is greater than HIGH.

---

### Diagram

Maximum value delimiter with hysteresis



HYST\_\*\*\*: Indicator for maximum value delimiter with hysteresis

---

---

## INDLIM\_\*\*\*: Indicator signal for delimiters with hysteresis

20

---

### Overview

#### Introduction

This chapter describes the INDLIM\_\*\*\* block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	72
Representation	72
Detailed description	73

INDLIM\_\*\*\*: Indicator signal for delimiters with hysteresis

---

## Brief description

---

### Function description

The function block monitors the input variable X for violation of the upper threshold and violation of the lower threshold.

**Note:** If only the upper threshold should be monitored as well, use the HYST function block.

It is possible to process data types of the ANY\_NUM group.  
The data types of all input values must be identical. A specific function is available to process each of the different data types.  
It is possible to project EN and ENO as additional parameters.

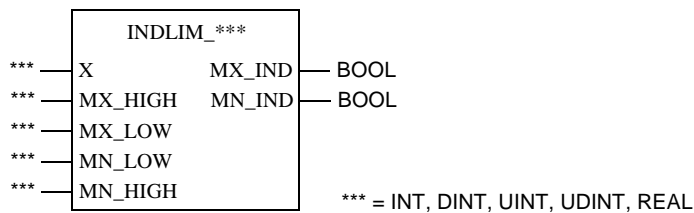
---

## Representation

---

### Symbol

Block representation:



### Parameter description

Block parameter description:

Parameter	Data type	Meaning
X	INT, DINT, UINT, UDINT, REAL	Input variable
MX_HIGH	INT, DINT, UINT, UDINT, REAL	Maximum upper limit
MX_LOW	INT, DINT, UINT, UDINT, REAL	Minimum lower limit
MN_LOW	INT, DINT, UINT, UDINT, REAL	Minimum lower limit
MN_HIGH	INT, DINT, UINT, UDINT, REAL	Minimum upper limit
MX_IND	BOOL	Display: reached upper limit
MN_IND	BOOL	Display: reached lower limit

---

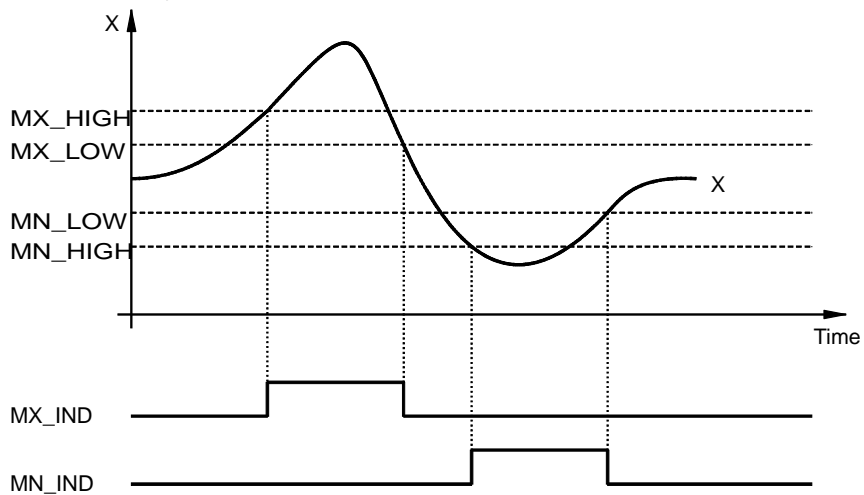
## Detailed description

**Parameter description** If X exceeds the upper limit MX\_HIGH, the function block reports this condition with MX\_IND = 1.  
If, subsequently, X is less than the limit MX\_LOW, MX\_IND will be reset to "0".

**Function** Function description:  
 $MX\_IND_i = 1$ , if  $X > MX\_HIGH$   
 $MX\_IND_i = 0$ , if  $X < MX\_LOW$   
 otherwise  
 $MX\_IND_i = MX\_IND_{(i-1)}$   
 If X is less than the lower limit MN\_HIGH, the function block reports this condition with MN\_IND = 1.  
 If, subsequently, X exceeds the limit MN\_LOW, MN\_IND is reset to "0".

**Function** Function description:  
 $MX\_IND_i = 1$ , if  $X < MX\_HIGH$   
 $MX\_IND_i = 0$ , if  $X > MX\_LOW$   
 otherwise  
 $MX\_IND_i = MX\_IND_{(i-1)}$   
 If MX\_LOW is greater than MX\_HIGH, there will be no hysteresis, and MX\_IND displays the fact that X is greater than MX\_HIGH.  
 If MN\_HIGH is greater than MN\_LOW, there will be no hysteresis, and MN\_IND displays the fact that X is smaller than MN\_HIGH.

**Diagram** Delimiters with hysteresis INDLIM



INDLIM\_\*\*\*: Indicator signal for delimiters with hysteresis

---

---

## INT\_TO\_BCD: Conversion from INT to 16 Bit BCD

21

---

### Overview

#### Introduction

This chapter describes the INT\_TO\_BCD block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	76
Representation	76

---

## Brief description

---

### Function description

This function converts an INT data type input value into a 16 Bit BCD output value (8-4-2-1-Code).

The valid input value range is 0 9 999 (Dec.).

If no valid BCD coded value is created at input, a runtime error is reported and the input value passes unchanged to the output.

EN and ENO can be configured as additional parameters.

---

### Example

Example of an INT -> BCD conversion

	Data type	Dec. value	Hex. value (= BCD-value)
<b>Input value</b>	INT	9 753	-
<b>Output value</b>	INT	-26 797	9753

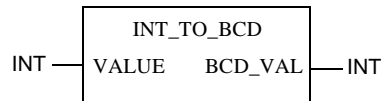
---

## Representation

---

### Symbol

Block representation:



### Parameter description

Block parameter description:

Parameter	Data type	Meaning
VALUE	INT	INT input value (valid value range 0 - 9 999)
BCD_VAL	INT	16 Bit BCD output value

---

---

## INT\_TO\_DBCD: Conversion from INT to 32 Bit BCD

22

---

### Overview

#### Introduction

This chapter describes the INT\_TO\_DBCD block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	78
Representation	78

---

## Brief description

---

### Function description

This function converts an INT data type input value into a 32 Bit BCD output value (8-4-2-1-Code).  
 The valid input value range is 0 32 767 (Dec.).  
 If no valid value is created at input, a runtime error is reported and the input value passes unchanged to the output.  
 EN and ENO can be configured as additional parameters.

---

### Example

Example of an INT -> DBCD conversion

	Data type	Dec. value	Hex. value (= BCD-value)
<b>Input value</b>	INT	13 579	-
<b>Output value</b>	DINT	79 225	0001 3579

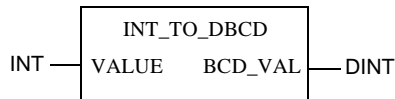
---

## Representation

---

### Symbol

Block representation:



### Parameter description

Block parameter description:

Parameter	Data type	Meaning
VALUE	INT	INT input value (valid value range 0 - 32767)
BCD_VAL	DINT	32 Bit BCD output value

---

---

## LIMIT\_IND\_\*\*\*: Limit with indicator

23

---

### Overview

#### Introduction

This chapter describes the LIMIT\_IND\_\*\*\* block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	80
Representation	80

## Brief description

---

### Function description

This function block transfers the unchanged input value (IN) to the output OUT, if the input value is not less than the minimum value (MN) and does not exceed the maximum value (MX). If the input value (IN) is less than the minimum value (MN), the minimum value will be transferred to the output. If the input value (IN) exceeds the maximum value (MX), the maximum value is transferred to the output. Furthermore, a signal indicates the violation of a minimum or maximum value. If the value at input IN is less than the value at input MN, output MN\_IND becomes "1". If the value at input IN exceeds the value at input MX, the output MX\_IND becomes "1".

Data types of the ANY\_ELEM group can be processed.

The data types of the input values MN, IN, MX as well as of the output value OUT must be identical. A specific function is available to process each of the different data types.

It is possible to project EN and ENO as additional parameters.

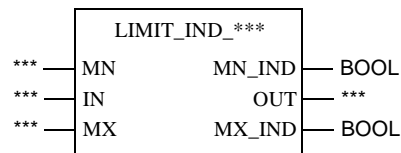
---

## Representation

---

### Symbol

Block representation:



\*\*\* = INT, DINT, UINT, UDINT, REAL, TIME,  
BOOL, BYTE, WORD

---

### Formula

Block formula:

$OUT = IN$ , if  $(IN \leq MX) \ \& \ IN \geq MN$

$OUT = MN$ , if  $(IN < MN)$

$OUT = MX$ , if  $(IN > MX)$

$MN\_IND = 0$ , if  $IN \geq MN$

$MN\_IND = 1$ , if  $IN < MN$

$MX\_IND = 0$ , if  $IN \leq MX$

$MX\_IND = 1$ , if  $IN > MX$

---

**Parameter  
description**

Block parameter description:

Parameter	Data type	Meaning
MN	INT, DINT, UINT, UDINT, REAL, TIME, BOOL, BYTE, WORD	Limit of minimum value
IN	INT, DINT, UINT, UDINT, REAL, TIME, BOOL, BYTE, WORD	Input
MX	INT, DINT, UINT, UDINT, REAL, TIME, BOOL, BYTE, WORD	Limit of maximum value
MN_IND	BOOL	Display of minimum value violation
OUT	INT, DINT, UINT, UDINT, REAL, TIME, BOOL, BYTE, WORD	Output
MX_IND	BOOL	Display of maximum value violation

LIMIT\_IND\_\*\*\*: Limit with indicator

---

---

## LOOKUP\_TABLE1: Traverse progression with 1st degree interpolation

24

---

### Overview

#### Introduction

This chapter describes the LOOKUP\_TABLE1 block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	84
Representation	84
Detailed description	85

## Brief description

---

### Function description

This function block linearizes characteristic curves by means of interpolation. The function block works with variable width. The number of XiYi inputs can be increased to 30 by modifying the size of the block frame vertically. This corresponds to a maximum of 15 support point pairs. The number of inputs must be even. The X values must be in ascending order. It is possible to project EN and ENO as additional parameters.

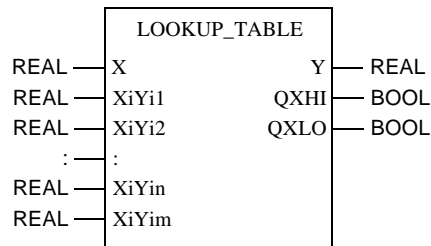
---

## Representation

---

### Symbol

Block representation:



### Parameter description

Block parameter description:

Parameter	Data type	Meaning
XiYi1	REAL	X-coordinate 1st support point
XiYi2	REAL	Y-coordinate 1st support point
XiYin	REAL	X-coordinate mth support point
XiYim	REAL	Y-coordinate mth support point
X	REAL	Input variable
Y	REAL	Output variable
QXHI	BOOL	Display: $X > X_m$
QXLO	BOOL	Indicator signal $X < X_1$

---

## Detailed description

### Parameter description

Each two sequential inputs ( $X_i Y_i$ ) represent a support point pair. The first input  $X_i Y_i$  corresponds to  $X_1$ , the next one to  $Y_1$ , the one after that to  $X_2$ , etc.

For all types of input value in  $X$  found between these support points, the corresponding  $Y$  output value is interpolated, while the traverse progression between the support points is viewed linearly.

For  $X < X_1$   $Y$  is =  $Y_1$

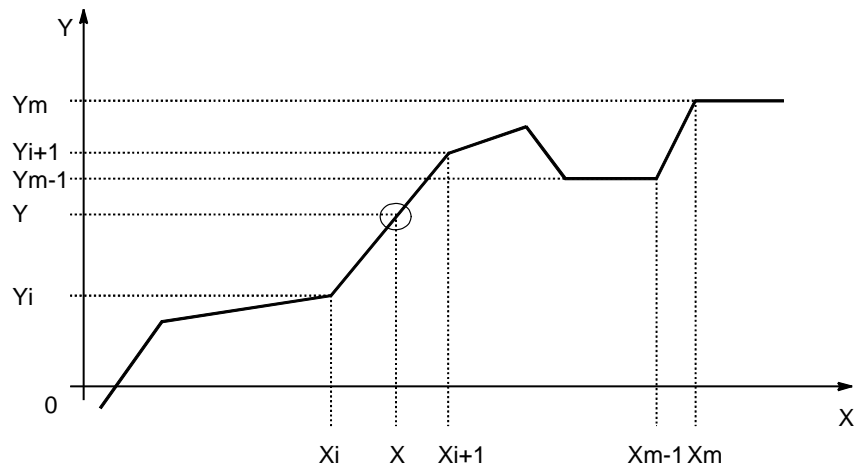
For  $X > X_m$  is  $Y = Y_m$

If the value at input  $X$  is higher than the value of the last support point  $X_m$ , the output QXHI becomes "1".

If the value at input  $X$  is less than the value of the first support point  $X_1$ , the output QXLO becomes "1".

### Principle of interpolation

Traverse progression with 1st degree interpolation



**Interpolation**

The following algorithm applies to a Point y:

$$Y = Y_i + \frac{Y_{(j+1)} - Y_j}{X_{(j+1)} - X_j} \times (X - X_i)$$

for  $X_i \leq X \leq X_{i+1}$  and  $i = 1 \dots (m-1)$

Assuming:  $X_1 \leq X_2 \leq \dots \leq X_i \leq X_{i+1} \leq \dots \leq X_{m-1} \leq X_m$

The X values must be in ascending order.

Two consecutive X values can be identical. This could cause a discrete curve progression.

In this instance, the special case applies:

$$Y = 0.5 \times (Y_i + Y_{i+1})$$

for

$X_i = X = X_{i+1}$  und  $i = 1 \dots (m-1)$

---

---

## NEG\_\*\*\*: Negation

25

---

### Overview

#### Introduction

This chapter describes the NEG\_\*\*\* block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	88
Representation	88
Runtime error	89

## Brief description

---

### Function description

The Function negates the input value and delivers the result at the OUT output. Data types of the ANY\_NUM group can be processed. The negation causes a sign reversal, e.g.  
 6 -> -6  
 -4 -> 4

**Note:** When the data types are processed DINT and INT it is not possible to convert very long negative values into positive ones. However, the ENO output is not set to 0 when this error occurs.

**Note:** When the data types are processed UDINT and UINT there is always an Error message.

The data types of the input and output values must be identical. A specific function is available to process each of the different data types. It is possible to project EN and ENO as additional parameters.

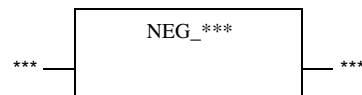
---

## Representation

---

### Symbol

Block representation:



\*\*\* = INT, DINT, UINT, UDINT, REAL

---

### Parameter description

Block parameter description:

Parameter	Data type	Meaning
IN	INT, DINT, UINT, UDINT, REAL	Input
OUT	INT, DINT, UINT, UDINT, REAL	Negated output

---

## Runtime error

---

### Runtime error

A violation of the value range at the input during the execution of the function will cause an error message to appear.

An Error message appears, when

- the value range of the input is exceeded or
  - an input value of the data type UDINT or UINT is to be converted.
-

NEG\_\*\*\*: Negation

---

---

## REAL\_AS\_WORD: Type conversion

26

---

### Overview

#### Introduction

This chapter describes the REAL\_AS\_WORD block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	92
Representation	92

## Brief description

---

### Function description

This function block converts one input word from Data type REAL to 2 output values of data type WORD.  
 The individual words of the REAL input are assigned to the outputs according to the output names.  
 It is possible to project EN and ENO as additional parameters.

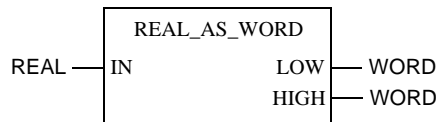
---

## Representation

---

### Symbol

Block representation:



### Parameter description

Block parameter description:

Parameter	Data type	Meaning
IN	REAL	Input
LOW	WORD	less significant word
HIGH	WORD	more significant word

---

---

## SAH: Detecting and holding with rising edge

27

---

### Overview

#### Introduction

This chapter describes the SAH block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	94
Representation	94

---

## Brief description

---

### Function description

The function block transfers the input value PV to the OUT output when first called up. With a rising edge (0 to 1) at input CLK, the input value IN is transferred to the OUT output. This value remains at the output until the next rising edge causes a new value to be loaded from IN to OUT.

The data types of the input values IN, PV and of the output value OUT must be identical.

The parameters EN and ENO can additionally be projected.

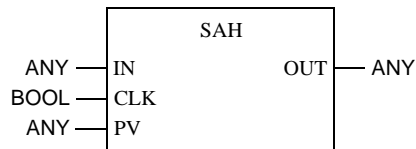
---

## Representation

---

### Symbol

Block representation:



### Parameter description

Block parameter description:

Parameter	Data type	Meaning
IN	ANY	Input value
CLK	BOOL	Clock input
PV	ANY	Preset value
OUT	ANY	Output value

---

---

## SIGN\_\*\*\*: Sign evaluation

28

---

### Overview

#### Introduction

This chapter describes the SIGN\_\*\*\* block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	96
Representation	96

## Brief description

---

### Function description

The function is used to detect negative signs.  
Data types of the ANY\_NUM group can be processed.  
With a value  $\geq 0$  at the input, the output becomes "0". With a value  $< 0$  at the input, the output becomes "1".

**Note:** Different processing of REAL and INT values results in the following behavior for signed 0 (+/-0):  
-0.0 -> SIGN\_REAL -> 1  
-0 -> SIGN\_INT/DINT -> 0  
+0.0 -> SIGN\_REAL -> 0  
+0 -> SIGN\_INT/DINT -> 0

A specific function is available to process each of the different data types.  
EN and ENO can be configured as additional parameters.

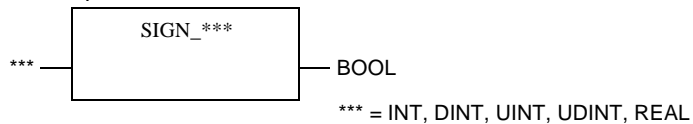
---

## Representation

---

### Symbol

Block representation:



### Formula

Block formula:  
OUT = 1, if IN < 0  
OUT = 0, if IN  $\geq 0$

**Note:** Different processing of REAL and INT values results in the following behavior for signed 0 (+/-0):  
-0.0 -> SIGN\_REAL -> 1  
-0 -> SIGN\_INT/DINT -> 0  
+0.0 -> SIGN\_REAL -> 0  
+0 -> SIGN\_INT/DINT -> 0

---

**Parameter  
description**

Block parameter description:

Parameter	Data type	Meaning
IN	INT, DINT, UINT, UDINT, REAL	Signed input
OUT	BOOL	Sign evaluation

---

SIGN\_\*\*\*: Sign evaluation

---

---

## TIME\_AS\_WORD: Type conversion

29

---

### Overview

#### Introduction

This chapter describes the TIME\_AS\_WORD block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	100
Representation	100

## Brief description

---

### Function description

The function block converts one input word from Data type TIME to 2 output values of data type WORD.  
 The individual words of the TIME input are assigned to the outputs according to the output names.  
 It is possible to project EN and ENO as additional parameters.

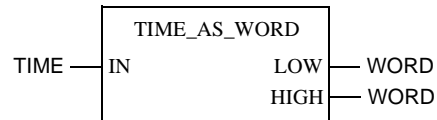
---

## Representation

---

### Symbol

Block representation:



### Parameter description

Block parameter description:

Parameter	Data type	Meaning
IN	TIME	Input
LOW	WORD	less significant word
HIGH	WORD	more significant word

---

---

## TOF\_P: Off Delay with Pause

30

---

### Overview

#### Introduction

This chapter describes the TOF\_P block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	102
Representation	102
Detailed description	103

---

## Brief description

### Function description

The function block is used as Off delay.

A 0 -> 1 edge on input IN triggers a reset.

A 1 -> 0 edge on input IN starts the timer function.

If the elapsed time (output ET) reaches the value defined on input PT, output Q is set to "0".

When the function block is first called the initial state of ET is "0".

With a "1" signal on input PAUSE, the timer is stopped and all values are frozen. If the input PAUSE becomes "0" again, the timer continues.

PAUSE has the highest priority. With a 1 -> 0 edge on input PAUSE, the block is no longer processed in this cycle. The block continues processing in the next cycle.

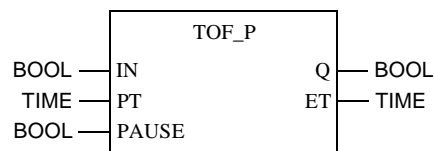
EN and ENO can be configured as additional parameters.

**Note:** Input EN cannot be used as pause function for the function block. The elapsed time is still measured even when input EN becomes "0". If input EN becomes "1" again, output ET is updated and executes a jump. However, if PAUSE is "1" when EN becomes "0", the pause is continued after EN becomes "1" until PAUSE becomes "0". In this case, there is no jump on output ET.

## Representation

### Symbol

Block representation:



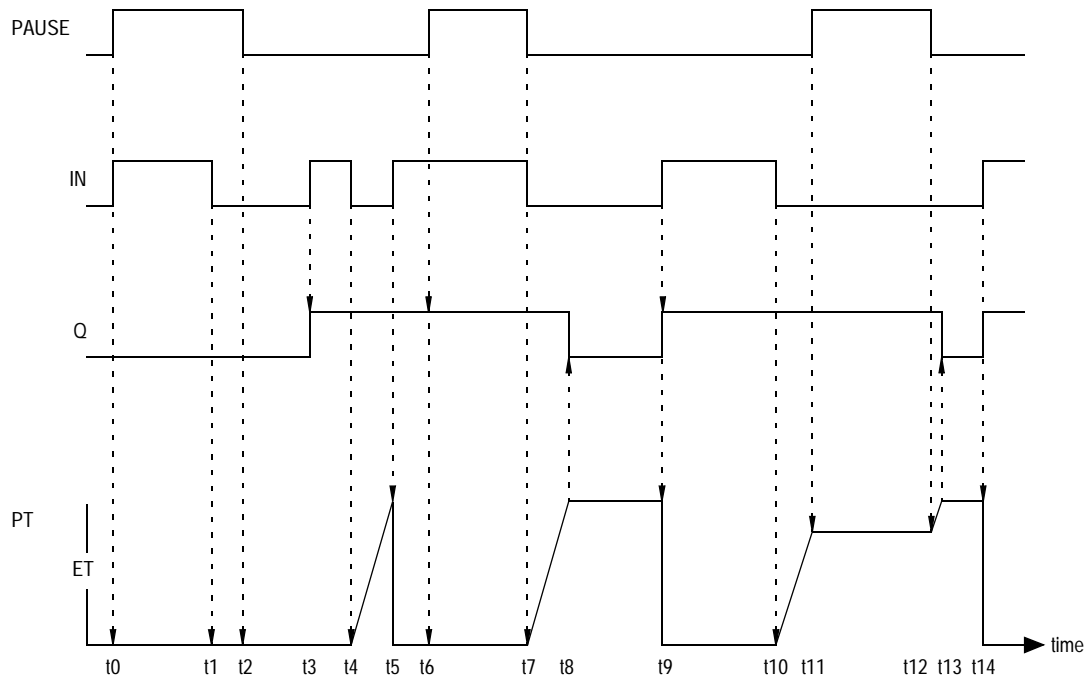
### Parameter description

Block parameter description:

Parameter	Data type	Meaning
IN	BOOL	0 -> 1: Reset 1 -> 0: Start timer
PT	TIME	Preset delay time
PAUSE	BOOL	1: The timer values are frozen.
Q	BOOL	Output
ET	TIME	Elapsed time

## Detailed description

**Timing diagram** Representation of the Off delay TOF\_P:



- t0** If IN becomes "1" and PAUSE is "1", Q remains "0" and the internal time is not started (PAUSE has priority above IN).
- t1** If IN becomes "0" while PAUSE is "1", the block remains inactive.
- t2** If PAUSE becomes "0" while IN is "0", the block remains inactive.
- t3** If IN becomes "1", Q becomes "1".
- t4** If IN becomes "0", the internal time (ET) is started.
- t5** If IN becomes "1" before the internal time has reached the value of PT, the internal time is reset without Q becoming "0".
- t6** If PAUSE becomes "1", Q remains "1" (the block is inactive).
- t7** If IN and PAUSE become "0", the internal time is started in the next cycle.
- t8** If the internal time reaches the value of PT, Q becomes "0".
- t9** If IN becomes "1", Q becomes "1" and the internal time is reset.
- t10** If IN becomes "0", the internal time is started.

TOF\_P: Off Delay with Pause

---

**t11** If PAUSE becomes "1" before the internal time has reached the value of PT, the internal time is stopped without Q becoming "0".

**t12** If PAUSE becomes "0", the internal time (ET) continues.

**t13** If the internal time reaches the value of PT, Q becomes "0".

**t14** If IN becomes "1", Q becomes "1" and the internal time is reset.

---

---

## TON\_P: On Delay with Pause

31

---

### Overview

#### Introduction

This chapter describes the TON\_P block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	106
Representation	106
Detailed description	107

## Brief description

### Function description

The function block is used as On delay.

A 1 -> 0 edge on input IN triggers a reset.

A 0 -> 1 edge on input IN starts the timer function.

If the elapsed time (output ET) reaches the value defined on input PT, output Q is set to "1".

When the function block is first called the initial state of ET is "0".

With a "1" signal on input PAUSE, the timer is stopped and all values are frozen. If the input PAUSE becomes "0" again, the timer continues.

PAUSE has the highest priority. With a 1 -> 0 edge on input PAUSE, the block is no longer processed in this cycle. The block continues processing in the next cycle.

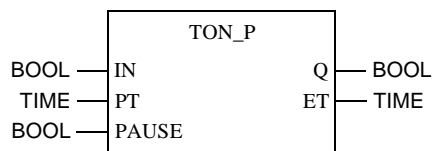
EN and ENO can be configured as additional parameters.

**Note:** Input EN cannot be used as pause function for the function block. The elapsed time is still measured even when input EN becomes "0". If input EN becomes "1" again, output ET is updated and executes a jump. However, if PAUSE is "1" when EN becomes "0", the pause is continued after EN becomes "1" until PAUSE becomes "0". In this case, there is no jump on output ET.

## Representation

### Symbol

Block representation:



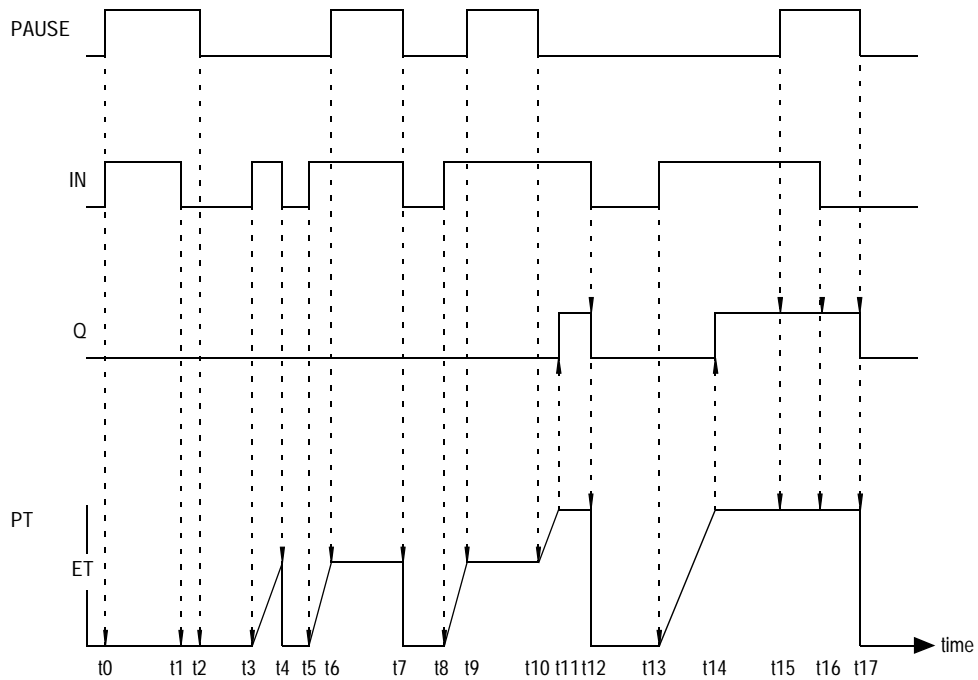
### Parameter description

Block parameter description:

Parameter	Data type	Meaning
IN	BOOL	0 -> 1: Start timer 1 -> 0: Reset
PT	TIME	Preset delay time
PAUSE	BOOL	1: The timer values are frozen.
Q	BOOL	Output
ET	TIME	Elapsed time

## Detailed description

**Timing diagram** Representation of the On delay TON\_P:



- t0** If IN becomes "1" and PAUSE is "1", Q remains "0" and the internal time is not started (PAUSE has priority above IN).
- t1** If IN becomes "0" while PAUSE is "1", the block remains inactive.
- t2** If PAUSE becomes "0" while IN is "0", the block remains inactive.
- t3** If IN becomes "1", the internal time (ET) is started.
- t4** If IN becomes "0" before the internal time has reached the value of PT, the internal time is reset without Q becoming "1".
- t5** If IN becomes "1", the internal time (ET) is started.
- t6** If PAUSE becomes "1" before the internal time has reached the value of PT, the internal time is stopped without Q becoming "1".
- t7** If IN and PAUSE become "0", the internal time is reset in the next cycle without Q becoming "1".
- t8** If IN becomes "1", the internal time (ET) is started.

- t9** If PAUSE becomes "1" before the internal time has reached the value of PT, the internal time is stopped without Q becoming "1".
  - t10** If PAUSE becomes "0", the internal time (ET) continues.
  - t11** If the internal time reaches the value of PT, Q becomes "1".
  - t12** If IN becomes "0", Q becomes "0" and the internal time is reset.
  - t13** If IN becomes "1", the internal time (ET) is started.
  - t14** If the internal time reaches the value of PT, Q becomes "1".
  - t15** If PAUSE becomes "1", Q remains "1".
  - t16** If IN becomes "0" while PAUSE is "1", Q remains "1" and ET is not reset (the block is inactive).
  - t17** If PAUSE becomes "0" while IN is "0", Q becomes "0" in the next cycle and the internal time is reset.
-

---

## TRIGGER: Detection of all types of edges

32

---

### Overview

#### Introduction

This chapter describes the TRIGGER block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	110
Representation	110

TRIGGER: Detection of all types of edges

---

## Brief description

---

### Function description

The Function block recognizes all types of edges (1 -> 0 and 0 -> 1) at input CLK. Output EDGE becomes "1" if there is a transition from "0" to "1" or from "1" to "0" at CLK; otherwise it remains on "0".

If there is a transition from "0" to "1" at input CLK, output RISE becomes "1"; if there is a transition from "1" to "0" at input CLK, output FALL becomes "1"; if neither of these two instances occurs, both outputs become "0".

The parameters EN and ENO can additionally be projected.

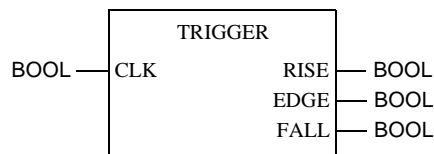
---

## Representation

---

### Symbol

Block representation:



### Parameter description

Block parameter description:

Parameter	Data type	Meaning
CLK	BOOL	Clock input
RISE	BOOL	Indicator signal of rising edge
EDGE	BOOL	Indicator of all types of edges
FALL	BOOL	Indicator signal of falling edge

---

---

## UDINT\_AS\_WORD: Type conversion

33

---

### Overview

#### Introduction

This chapter describes the UDINT\_AS\_WORD block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	112
Representation	112

---

## Brief description

---

### Function description

The function block converts one input word from Data type UDINT to 2 output values of data type WORD.  
The individual words of the UDINT input are assigned to the outputs according to the output names.  
It is possible to project EN and ENO as additional parameters.

---

## Representation

---

### Symbol

Block representation:



### Parameter description

Block parameter description:

Parameter	Data type	Meaning
IN	UDINT	Input
LOW	WORD	less significant word
HIGH	WORD	more significant word

---

---

## WORD\_AS\_BYTE: Type conversion

34

---

### Overview

#### Introduction

This chapter describes the WORD\_AS\_BYTE block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	114
Representation	114

## Brief description

---

### Function description

The function block converts one input word from Data type WORD to two output values of data type BYTE.  
The individual bytes of the word at the input are assigned to the outputs according to the output names.  
The parameters EN and ENO can additionally be projected.

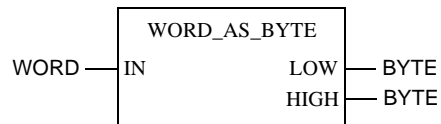
---

## Representation

---

### Symbol

Block representation:



### Parameter description

Block parameter description:

Parameter	Data type	Meaning
IN	WORD	Input
LOW	BYTE	less significant byte
HIGH	BYTE	more significant byte

---

---

## WORD\_AS\_DINT: Type conversion

35

---

### Overview

#### Introduction

This chapter describes the WORD\_AS\_DINT block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	116
Representation	116

## Brief description

---

### Function description

The Function converts 2 input words from Data type WORD to an output value of data type DINT.

The input values are assigned to the word at the output according to the input names.

The parameters EN and ENO can additionally be projected.

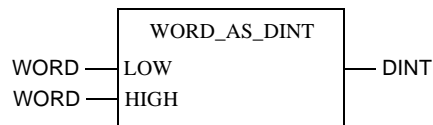
---

## Representation

---

### Symbol

Block representation:



### Formula

Block formula:

$$OUT = \{HIGH, LOW\}$$

---

### Parameter description

Block parameter description:

Parameter	Data type	Meaning
LOW	WORD	less significant word
HIGH	WORD	more significant word
OUT	DINT	Output value

---

---

## WORD\_AS\_REAL: Type conversion

36

---

### Overview

#### Introduction

This chapter describes the WORD\_AS\_REAL block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	118
Representation	118

## Brief description

---

### Function description

The Function converts 2 input words from Data type WORD to an output value of data type REAL.  
The input values are assigned to the word at the output according to the input names.  
It is possible to project EN and ENO as additional parameters.

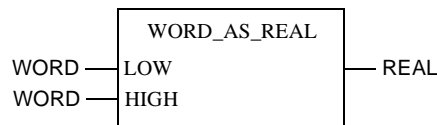
---

## Representation

---

### Symbol

Block representation:



### Formula

Block formula:

$$\text{OUT} = \{\text{HIGH}, \text{LOW}\}$$

---

### Parameter description

Block parameter description:

Parameter	Data type	Meaning
LOW	WORD	less significant word
HIGH	WORD	more significant word
OUT	REAL	Output value

---

---

## WORD\_AS\_TIME: Type conversion

37

---

### Overview

#### Introduction

This chapter describes the WORD\_AS\_TIME block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	120
Representation	120

## Brief description

---

### Function description

The Function converts 2 input words from Data type WORD to an output value of data type TIME.  
 The input values are assigned to the word at the output according to the input names.  
 The parameters EN and ENO can additionally be projected.

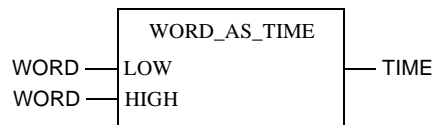
---

## Representation

---

### Symbol

Block representation:



### Formula

Block formula:

$$OUT = \{HIGH, LOW\}$$


---

### Parameter description

Block parameter description:

Parameter	Data type	Meaning
LOW	WORD	less significant word
HIGH	WORD	more significant word
OUT	TIME	Output value

---

---

## WORD\_AS\_UDINT: Type conversion

38

---

### Overview

#### Introduction

This chapter describes the WORD\_AS\_UDINT block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	122
Representation	122

---

## Brief description

---

### Function description

The Function converts 2 input words from Data type WORD to an output value of data type UDINT.  
The input values are assigned to the word at the output according to the input names.  
It is possible to project EN and ENO as additional parameters.

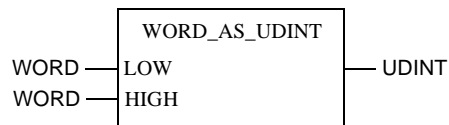
---

## Representation

---

### Symbol

Block representation:



### Formula

Block formula:

$$OUT = \{HIGH, LOW\}$$

---

### Parameter description

Block parameter description:

Parameter	Data type	Meaning
LOW	WORD	less significant word
HIGH	WORD	more significant word
OUT	UDINT	Output value

---

---

## WORD\_TO\_BIT: Type conversion

39

---

### Overview

#### Introduction

This chapter describes the WORD\_TO\_BIT block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	124
Representation	125

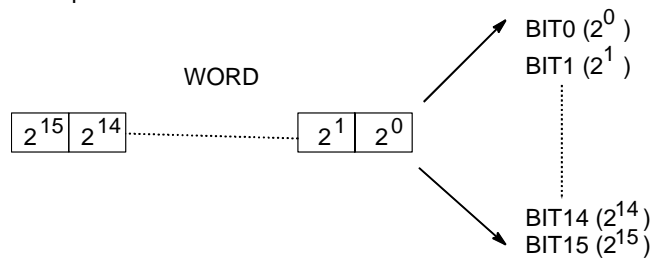
---

## Brief description

---

### Function description

The function block converts one input word from Data type WORD to 16 output values of data type BOOL. The individual bits of the word at the input are assigned to the outputs according to the output names.



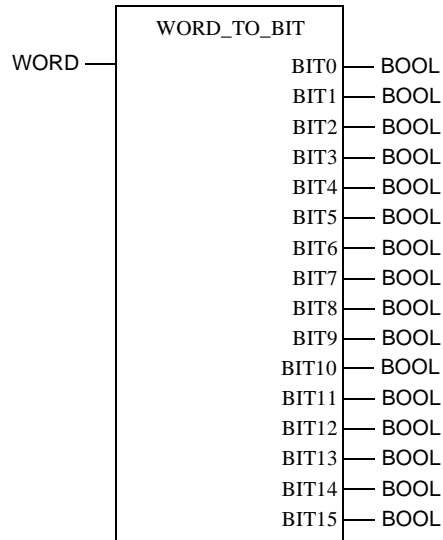
The parameters EN and ENO can additionally be projected.

---

## Representation

### Symbol

Block representation:



### Parameter description

Block parameter description:

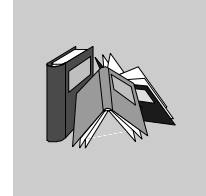
Parameter	Data type	Meaning
IN	WORD	Input
BIT0	BOOL	Output BIT0
BIT1	BOOL	Output BIT1
:	:	:
BIT15	BOOL	Output BIT15

WORD\_TO\_BIT: Type conversion

---

---

## Glossary



### A

- Active window** The window, which is currently selected. Only one window can be active at any one given time. When a window is active, the heading changes color, in order to distinguish it from other windows. Unselected windows are inactive.
- Actual parameter** Currently connected Input/Output parameters.
- Addresses** (Direct) addresses are memory areas on the PLC. These are found in the State RAM and can be assigned input/output modules.  
The display/input of direct addresses is possible in the following formats:
- Standard format (400001)
  - Separator format (4:00001)
  - Compact format (4:1)
  - IEC format (QW1)
- ANL\_IN** ANL\_IN stands for the data type "Analog Input" and is used for processing analog values. The 3x References of the configured analog input module, which is specified in the I/O component list is automatically assigned the data type and should therefore only be occupied by Unlocated variables.
- ANL\_OUT** ANL\_OUT stands for the data type "Analog Output" and is used for processing analog values. The 4x-References of the configured analog output module, which is specified in the I/O component list is automatically assigned the data type and should therefore only be occupied by Unlocated variables.
- ANY** In the existing version "ANY" covers the elementary data types BOOL, BYTE, DINT, INT, REAL, UDINT, UINT, TIME and WORD and therefore derived data types.

<b>ANY_BIT</b>	In the existing version, "ANY_BIT" covers the data types BOOL, BYTE and WORD.
<b>ANY_ELEM</b>	In the existing version "ANY_ELEM" covers the elementary data types BOOL, BYTE, DINT, INT, REAL, UDINT, UINT, TIME and WORD.
<b>ANY_INT</b>	In the existing version, "ANY_INT" covers the data types DINT, INT, UDINT and UINT.
<b>ANY_NUM</b>	In the existing version, "ANY_NUM" covers the data types DINT, INT, REAL, UDINT and UINT.
<b>ANY_REAL</b>	In the existing version "ANY_REAL" covers the data type REAL.
<b>Application window</b>	The window, which contains the working area, the menu bar and the tool bar for the application. The name of the application appears in the heading. An application window can contain several document windows. In Concept the application window corresponds to a Project.
<b>Argument</b>	Synonymous with Actual parameters.
<b>ASCII mode</b>	American Standard Code for Information Interchange. The ASCII mode is used for communication with various host devices. ASCII works with 7 data bits.
<b>Atrium</b>	The PC based controller is located on a standard AT board, and can be operated within a host computer in an ISA bus slot. The module occupies a motherboard (requires SA85 driver) with two slots for PC104 daughter boards. From this, a PC104 daughter board is used as a CPU and the others for INTERBUS control.

---

**B**

<b>Back up data file (Concept EFB)</b>	The back up file is a copy of the last Source files. The name of this back up file is "backup??.c" (it is accepted that there are no more than 100 copies of the source files. The first back up file is called "backup00.c". If changes have been made on the Definition file, which do not create any changes to the interface in the EFB, there is no need to create a back up file by editing the source files ( <b>Objects</b> → <b>Source</b> ). If a back up file can be assigned, the name of the source file can be given.
--	---

<b>Base 16 literals</b>	<p>Base 16 literals function as the input of whole number values in the hexadecimal system. The base must be denoted by the prefix 16#. The values may not be preceded by signs (+/-). Single underline signs ( _ ) between figures are not significant.</p> <p>Example 16#F_F or 16#FF (decimal 255) 16#E_0 or 16#E0 (decimal 224)</p>
<b>Base 8 literal</b>	<p>Base 8 literals function as the input of whole number values in the octal system. The base must be denoted by the prefix 8#. The values may not be preceded by signs (+/-). Single underline signs ( _ ) between figures are not significant.</p> <p>Example 8#3_1111 or 8#377 (decimal 255) 8#34_1111 or 8#340 (decimal 224)</p>
<b>Basis 2 literals</b>	<p>Base 2 literals function as the input of whole number values in the dual system. The base must be denoted by the prefix 2#. The values may not be preceded by signs (+/-). Single underline signs ( _ ) between figures are not significant.</p> <p>Example 2#1111_1111 or 2#11111111 (decimal 255) 2#1110_1111 or 2#11100000 (decimal 224)</p>
<b>Binary connections</b>	<p>Connections between outputs and inputs of FFBs of data type BOOL.</p>
<b>Bit sequence</b>	<p>A data element, which is made up from one or more bits.</p>
<b>BOOL</b>	<p>BOOL stands for the data type "Boolean". The length of the data elements is 1 bit (in the memory contained in 1 byte). The range of values for variables of this type is 0 (FALSE) and 1 (TRUE).</p>
<b>Bridge</b>	<p>A bridge serves to connect networks. It enables communication between nodes on the two networks. Each network has its own token rotation sequence – the token is not deployed via bridges.</p>
<b>BYTE</b>	<p>BYTE stands for the data type "Bit sequence 8". The input appears as Base 2 literal, Base 8 literal or Base 16 literal. The length of the data element is 8 bit. A numerical range of values cannot be assigned to this data type.</p>

**C**

<b>Cache</b>	The cache is a temporary memory for cut or copied objects. These objects can be inserted into sections. The old content in the cache is overwritten for each new Cut or Copy.
<b>Call up</b>	The operation, by which the execution of an operation is initiated.
<b>Coil</b>	A coil is a LD element, which transfers (without alteration) the status of the horizontal link on the left side to the horizontal link on the right side. In this way, the status is saved in the associated Variable/ direct address.
<b>Compact format (4:1)</b>	The first figure (the Reference) is separated from the following address with a colon (:), where the leading zero are not entered in the address.
<b>Connection</b>	A check or flow of data connection between graphic objects (e.g. steps in the SFC editor, Function blocks in the FBD editor) within a section, is graphically shown as a line.
<b>Constants</b>	Constants are Unlocated variables, which are assigned a value that cannot be altered from the program logic (write protected).
<b>Contact</b>	A contact is a LD element, which transfers a horizontal connection status onto the right side. This status is from the Boolean AND- operation of the horizontal connection status on the left side with the status of the associated Variables/direct Address. A contact does not alter the value of the associated variables/direct address.

---

**D**

- Data transfer settings** Settings, which determine how information from the programming device is transferred to the PLC.
- Data types** The overview shows the hierarchy of data types, as they are used with inputs and outputs of Functions and Function blocks. Generic data types are denoted by the prefix "ANY".
- ANY\_ELEM
    - ANY\_NUM
      - ANY\_REAL (REAL)
      - ANY\_INT (DINT, INT, UDINT, UINT)
    - ANY\_BIT (BOOL, BYTE, WORD)
    - TIME
  - System data types (IEC extensions)
  - Derived (from "ANY" data types)
- DCP I/O station** With a Distributed Control Processor (D908) a remote network can be set up with a parent PLC. When using a D908 with remote PLC, the parent PLC views the remote PLC as a remote I/O station. The D908 and the remote PLC communicate via the system bus, which results in high performance, with minimum effect on the cycle time. The data exchange between the D908 and the parent PLC takes place at 1.5 Megabits per second via the remote I/O bus. A parent PLC can support up to 31 (Address 2-32) D908 processors.
- DDE (Dynamic Data Exchange)** The DDE interface enables a dynamic data exchange between two programs under Windows. The DDE interface can be used in the extended monitor to call up its own display applications. With this interface, the user (i.e. the DDE client) can not only read data from the extended monitor (DDE server), but also write data onto the PLC via the server. Data can therefore be altered directly in the PLC, while it monitors and analyzes the results. When using this interface, the user is able to make their own "Graphic-Tool", "Face Plate" or "Tuning Tool", and integrate this into the system. The tools can be written in any DDE supporting language, e.g. Visual Basic and Visual-C++. The tools are called up, when the one of the buttons in the dialog box extended monitor uses Concept Graphic Tool: Signals of a projection can be displayed as timing diagrams via the DDE connection between Concept and Concept Graphic Tool.

<b>Decentral Network (DIO)</b>	A remote programming in Modbus Plus network enables maximum data transfer performance and no specific requests on the links. The programming of a remote net is easy. To set up the net, no additional ladder diagram logic is needed. Via corresponding entries into the Peer Cop processor all data transfer requests are met.
<b>Declaration</b>	Mechanism for determining the definition of a Language element. A declaration normally covers the connection of an Identifier with a language element and the assignment of attributes such as Data types and algorithms.
<b>Definition data file (Concept EFB)</b>	The definition file contains general descriptive information about the selected FFB and its formal parameters.
<b>Derived data type</b>	Derived data types are types of data, which are derived from the Elementary data types and/or other derived data types. The definition of the derived data types appears in the data type editor in Concept. Distinctions are made between global data types and local data types.
<b>Derived Function Block (DFB)</b>	A derived function block represents the Call up of a derived function block type. Details of the graphic form of call up can be found in the definition " Function block (Item)". Contrary to calling up EFB types, calling up DFB types is denoted by double vertical lines on the left and right side of the rectangular block symbol. The body of a derived function block type is designed using FBD language, but only in the current version of the programming system. Other IEC languages cannot yet be used for defining DFB types, nor can derived functions be defined in the current version. Distinctions are made between local and global DFBs.
<b>DINT</b>	DINT stands for the data type "double integer". The input appears as Integer literal, Base 2 literal, Base 8 literal or Base 16 literal. The length of the data element is 32 bit. The range of values for variables of this data type is from $-2 \exp(31)$ to $2 \exp(31) - 1$ .
<b>Direct display</b>	A method of displaying variables in the PLC program, from which the assignment of configured memory can be directly and indirectly derived from the physical memory.
<b>Document window</b>	A window within an Application window. Several document windows can be opened at the same time in an application window. However, only one document window can be active. Document windows in Concept are, for example, sections, the message window, the reference data editor and the PLC configuration.
<b>Dummy</b>	An empty data file, which consists of a text header with general file information, i.e. author, date of creation, EFB identifier etc. The user must complete this dummy file with additional entries.

---

**DX Zoom** This property enables connection to a programming object to observe and, if necessary, change its data value.

---

**E**

**Elementary functions/function blocks (EFB)** Identifier for Functions or Function blocks, whose type definitions are not formulated in one of the IEC languages, i.e. whose bodies, for example, cannot be modified with the DFB Editor (Concept-DFB). EFB types are programmed in "C" and mounted via Libraries in precompiled form.

**EN/ENO (Enable / Error display)** If the value of EN is "0" when the FFB is called up, the algorithms defined by the FFB are not executed and all outputs contain the previous value. The value of ENO is automatically set to "0" in this case. If the value of EN is "1" when the FFB is called up, the algorithms defined by the FFB are executed. After the error free execution of the algorithms, the ENO value is automatically set to "1". If an error occurs during the execution of the algorithm, ENO is automatically set to "0". The output behavior of the FFB depends whether the FFBs are called up without EN/ENO or with EN=1. If the EN/ENO display is enabled, the EN input must be active. Otherwise, the FFB is not executed. The projection of EN and ENO is enabled/disabled in the block properties dialog box. The dialog box is called up via the menu commands **Objects** → **Properties...** or via a double click on the FFB.

**Error** When processing a FFB or a Step an error is detected (e.g. unauthorized input value or a time error), an error message appears, which can be viewed with the menu command **Online** → **Event viewer...** . With FFBs the ENO output is set to "0".

**Evaluation** The process, by which a value for a Function or for the outputs of a Function block during the Program execution is transmitted.

**Expression** Expressions consist of operators and operands.

---

**F**

**FFB (functions/function blocks)** Collective term for EFB (elementary functions/function blocks) and DFB (derived function blocks)

**Field variables** Variables, one of which is assigned, with the assistance of the key word ARRAY (field), a defined Derived data type. A field is a collection of data elements of the same Data type.

---

<b>FIR filter</b>	Finite Impulse Response Filter
<b>Formal parameters</b>	Input/Output parameters, which are used within the logic of a FFB and led out of the FFB as inputs/outputs.
<b>Function (FUNC)</b>	<p>A Program organization unit, which exactly supplies a data element when executing. A function has no internal status information. Multiple call ups of the same function with the same input parameter values always supply the same output values. Details of the graphic form of function call up can be found in the definition " Function block (Item)". In contrast to the call up of function blocks, the function call ups only have one unnamed output, whose name is the name of the function itself. In FBD each call up is denoted by a unique number over the graphic block; this number is automatically generated and cannot be altered.</p>
<b>Function block (item) (FB)</b>	<p>A function block is a Program organization unit, which correspondingly calculates the functionality values, defined in the function block type description, for the output and internal variables, when it is called up as a certain item. All output values and internal variables of a certain function block item remain as a call up of the function block until the next. Multiple call up of the same function block item with the same arguments (Input parameter values) supply generally supply the same output value(s).</p> <p>Each function block item is displayed graphically by a rectangular block symbol. The name of the function block type is located on the top center within the rectangle. The name of the function block item is located also at the top, but on the outside of the rectangle. An instance is automatically generated when creating, which can however be altered manually, if required. Inputs are displayed on the left side and outputs on the right of the block. The names of the formal input/output parameters are displayed within the rectangle in the corresponding places.</p> <p>The above description of the graphic presentation is principally applicable to Function call ups and to DFB call ups. Differences are described in the corresponding definitions.</p>
<b>Function block dialog (FBD)</b>	One or more sections, which contain graphically displayed networks from Functions, Function blocks and Connections.
<b>Function block type</b>	<p>A language element, consisting of: 1. the definition of a data structure, subdivided into input, output and internal variables, 2. A set of operations, which is used with the elements of the data structure, when a function block type instance is called up. This set of operations can be formulated either in one of the IEC languages (DFB type) or in "C" (EFB type). A function block type can be instanced (called up) several times.</p>

---

**Function counter** The function counter serves as a unique identifier for the function in a Program or DFB. The function counter cannot be edited and is automatically assigned. The function counter always has the structure: .n.m

n = Section number (number running)

m = Number of the FFB object in the section (number running)

---

**G**

**Generic data type** A Data type, which stands in for several other data types.

**Generic literal** If the Data type of a literal is not relevant, simply enter the value for the literal. In this case Concept automatically assigns the literal to a suitable data type.

**Global derived data types** Global Derived data types are available in every Concept project and are contained in the DFB directory directly under the Concept directory.

**Global DFBs** Global DFBs are available in every Concept project and are contained in the DFB directory directly under the Concept directory.

**Global macros** Global Macros are available in every Concept project and are contained in the DFB directory directly under the Concept directory.

**Groups (EFBs)** Some EFB libraries (e.g. the IEC library) are subdivided into groups. This facilitates the search for FFBs, especially in extensive libraries.

---

**I**

**I/O component list** The I/O and expert assemblies of the various CPUs are configured in the I/O component list.

**IEC 61131-3** International norm: Programmable controllers – part 3: Programming languages.

---

**IEC format (QW1)** In the place of the address stands an IEC identifier, followed by a five figure address:

- %0x12345 = %Q12345
- %1x12345 = %I12345
- %3x12345 = %IW12345
- %4x12345 = %QW12345

**IEC name conventions (identifier)** An identifier is a sequence of letters, figures, and underscores, which must start with a letter or underscores (e.g. name of a function block type, of an item or section). Letters from national sets of characters (e.g. ö, ü, é, ð) can be used, taken from project and DFB names. Underscores are significant in identifiers; e.g. "A\_BCD" and "AB\_CD" are interpreted as different identifiers. Several leading and multiple underscores are not authorized consecutively. Identifiers are not permitted to contain space characters. Upper and/or lower case is not significant; e.g. "ABCD" and "abcd" are interpreted as the same identifier. Identifiers are not permitted to be Key words.

**IIR filter** Infinite Impulse Response Filter

**Initial step (starting step)** The first step in a chain. In each chain, an initial step must be defined. The chain is started with the initial step when first called up.

**Initial value** The allocated value of one of the variables when starting the program. The value assignment appears in the form of a Literal.

**Input bits (1x references)** The 1/0 status of input bits is controlled via the process data, which reaches the CPU from an entry device.

<p><b>Note:</b> The x, which comes after the first figure of the reference type, represents a five figure storage location in the application data store, i.e. if the reference 100201 signifies an input bit in the address 201 of the State RAM.</p>
--

**Input parameters (Input)** When calling up a FFB the associated Argument is transferred.

**Input words (3x references)** An input word contains information, which come from an external source and are represented by a 16 bit figure. A 3x register can also contain 16 sequential input bits, which were read into the register in binary or BCD (binary coded decimal) format. Note: The x, which comes after the first figure of the reference type, represents a five figure storage location in the user data store, i.e. if the reference 300201 signifies a 16 bit input word in the address 201 of the State RAM.

**Instantiation** The generation of an Item.

---

<b>Instruction (IL)</b>	Instructions are "commands" of the IL programming language. Each operation begins on a new line and is succeeded by an operator (with modifier if needed) and, if necessary for each relevant operation, by one or more operands. If several operands are used, they are separated by commas. A tag can stand before the instruction, which is followed by a colon. The commentary must, if available, be the last element in the line.
<b>Instruction (LL984)</b>	When programming electric controllers, the task of implementing operational coded instructions in the form of picture objects, which are divided into recognizable contact forms, must be executed. The designed program objects are, on the user level, converted to computer useable OP codes during the loading process. The OP codes are deciphered in the CPU and processed by the controller's firmware functions so that the desired controller is implemented.
<b>Instruction list (IL)</b>	IL is a text language according to IEC 1131, in which operations, e.g. conditional/unconditional call up of Function blocks and Functions, conditional/unconditional jumps etc. are displayed through instructions.
<b>INT</b>	INT stands for the data type "whole number". The input appears as Integer literal, Base 2 literal, Base 8 literal or Base 16 literal. The length of the data element is 16 bit. The range of values for variables of this data type is from $-2 \exp (15)$ to $2 \exp (15) - 1$ .
<b>Integer literals</b>	Integer literals function as the input of whole number values in the decimal system. The values may be preceded by the signs (+/-). Single underline signs ( <u>  </u> ) between figures are not significant.  Example -12, 0, 123_456, +986
<b>INTERBUS (PCP)</b>	To use the INTERBUS PCP channel and the INTERBUS process data preprocessing (PDP), the new I/O station type INTERBUS (PCP) is led into the Concept configurator. This I/O station type is assigned fixed to the INTERBUS connection module 180-CRP-660-01. The 180-CRP-660-01 differs from the 180-CRP-660-00 only by a clearly larger I/O area in the state RAM of the controller.

**Item name** An Identifier, which belongs to a certain Function block item. The item name serves as a unique identifier for the function block in a program organization unit. The item name is automatically generated, but can be edited. The item name must be unique throughout the Program organization unit, and no distinction is made between upper/lower case. If the given name already exists, a warning is given and another name must be selected. The item name must conform to the IEC name conventions, otherwise an error message appears. The automatically generated instance name always has the structure: FBI\_n\_m

FBI = Function block item  
n = Section number (number running)  
m = Number of the FFB object in the section (number running)

---

**J**

**Jump** Element of the SFC language. Jumps are used to jump over areas of the chain.

---

**K**

**Key words** Key words are unique combinations of figures, which are used as special syntactic elements, as is defined in appendix B of the IEC 1131-3. All key words, which are used in the IEC 1131-3 and in Concept, are listed in appendix C of the IEC 1131-3. These listed keywords cannot be used for any other purpose, i.e. not as variable names, section names, item names etc.

---

**L**

**Ladder Diagram (LD)** Ladder Diagram is a graphic programming language according to IEC1131, which optically orientates itself to the "rung" of a relay ladder diagram.

---

---

<b>Ladder Logic 984 (LL)</b>	<p>In the terms Ladder Logic and Ladder Diagram, the word Ladder refers to execution. In contrast to a diagram, a ladder logic is used by engineers to draw up a circuit (with assistance from electrical symbols), which should chart the cycle of events and not the existing wires, which connect the parts together. A usual user interface for controlling the action by automated devices permits ladder logic interfaces, so that when implementing a control system, engineers do not have to learn any new programming languages, with which they are not conversant.</p> <p>The structure of the actual ladder logic enables electrical elements to be linked in a way that generates a control output, which is dependant upon a configured flow of power through the electrical objects used, which displays the previously demanded condition of a physical electric appliance.</p> <p>In simple form, the user interface is one of the video displays used by the PLC programming application, which establishes a vertical and horizontal grid, in which the programming objects are arranged. The logic is powered from the left side of the grid, and by connecting activated objects the electricity flows from left to right.</p>
<b>Landscape format</b>	<p>Landscape format means that the page is wider than it is long when looking at the printed text.</p>
<b>Language element</b>	<p>Each basic element in one of the IEC programming languages, e.g. a Step in SFC, a Function block item in FBD or the Start value of a variable.</p>
<b>Library</b>	<p>Collection of software objects, which are provided for reuse when programming new projects, or even when building new libraries. Examples are the Elementary function block types libraries.</p> <p>EFB libraries can be subdivided into Groups.</p>
<b>Literals</b>	<p>Literals serve to directly supply values to inputs of FFBs, transition conditions etc. These values cannot be overwritten by the program logic (write protected). In this way, generic and standardized literals are differentiated.</p> <p>Furthermore literals serve to assign a Constant a value or a Variable an Initial value. The input appears as Base 2 literal, Base 8 literal, Base 16 literal, Integer literal, Real literal or Real literal with exponent.</p>
<b>Local derived data types</b>	<p>Local derived data types are only available in a single Concept project and its local DFBs and are contained in the DFB directory under the project directory.</p>
<b>Local DFBs</b>	<p>Local DFBs are only available in a single Concept project and are contained in the DFB directory under the project directory.</p>
<b>Local link</b>	<p>The local network link is the network, which links the local nodes with other nodes either directly or via a bus amplifier.</p>
<b>Local macros</b>	<p>Local Macros are only available in a single Concept project and are contained in the DFB directory under the project directory.</p>

**Local network nodes** The local node is the one, which is projected evenly.

**Located variable** Located variables are assigned a state RAM address (reference addresses 0x, 1x, 3x, 4x). The value of these variables is saved in the state RAM and can be altered online with the reference data editor. These variables can be addressed by symbolic names or the reference addresses.

Collective PLC inputs and outputs are connected to the state RAM. The program access to the peripheral signals, which are connected to the PLC, appears only via located variables. PLC access from external sides via Modbus or Modbus plus interfaces, i.e. from visualizing systems, are likewise possible via located variables.

---

## M

**Macro** Macros are created with help from the software Concept DFB. Macros function to duplicate frequently used sections and networks (including the logic, variables, and variable declaration). Distinctions are made between local and global macros.

Macros have the following properties:

- Macros can only be created in the programming languages FBD and LD.
- Macros only contain one single section.
- Macros can contain any complex section.
- From a program technical point of view, there is no differentiation between an instanced macro, i.e. a macro inserted into a section, and a conventionally created macro.
- Calling up DFBs in a macro
- Variable declaration
- Use of macro-own data structures
- Automatic acceptance of the variables declared in the macro
- Initial value for variables
- Multiple instancing of a macro in the whole program with different variables
- The section name, the variable name and the data structure name can contain up to 10 different exchange markings (@0 to @9).

**MMI** Man Machine Interface

**Multi element variables** Variables, one of which is assigned a Derived data type defined with STRUCT or ARRAY. Distinctions are made between Field variables and structured variables.

---

**N**

<b>Network</b>	A network is the connection of devices to a common data path, which communicate with each other via a common protocol.
<b>Network node</b>	A node is a device with an address (164) on the Modbus Plus network.
<b>Node address</b>	The node address serves a unique identifier for the network in the routing path. The address is set directly on the node, e.g. with a rotary switch on the back of the module.

---

**O**

<b>Operand</b>	An operand is a Literal, a Variable, a Function call up or an Expression.
<b>Operator</b>	An operator is a symbol for an arithmetic or Boolean operation to be executed.
<b>Output parameters (Output)</b>	A parameter, with which the result(s) of the Evaluation of a FFB are returned.
<b>Output/discretes (0x references)</b>	An output/marker bit can be used to control real output data via an output unit of the control system, or to define one or more outputs in the state RAM. Note: The x, which comes after the first figure of the reference type, represents a five figure storage location in the application data store, i.e. if the reference 000201 signifies an output or marker bit in the address 201 of the State RAM.
<b>Output/marker words (4x references)</b>	An output/marker word can be used to save numerical data (binary or decimal) in the State RAM, or also to send data from the CPU to an output unit in the control system. Note: The x, which comes after the first figure of the reference type, represents a five figure storage location in the application data store, i.e. if the reference 400201 signifies a 16 bit output or marker word in the address 201 of the State RAM.

---

**P**

<b>Peer processor</b>	The peer processor processes the token run and the flow of data between the Modbus Plus network and the PLC application logic.
<b>PLC</b>	Programmable controller
<b>Program</b>	The uppermost Program organization unit. A program is closed and loaded onto a single PLC.
<b>Program cycle</b>	A program cycle consists of reading in the inputs, processing the program logic and the output of the outputs.
<b>Program organization unit</b>	A Function, a Function block, or a Program. This term can refer to either a Type or an Item.
<b>Programming device</b>	Hardware and software, which supports programming, configuring, testing, implementing and error searching in PLC applications as well as in remote system applications, to enable source documentation and archiving. The programming device could also be used for process visualization.
<b>Programming redundancy system (Hot Standby)</b>	A redundancy system consists of two identically configured PLC devices, which communicate with each other via redundancy processors. In the case of the primary PLC failing, the secondary PLC takes over the control checks. Under normal conditions the secondary PLC does not take over any controlling functions, but instead checks the status information, to detect mistakes.
<b>Project</b>	General identification of the uppermost level of a software tree structure, which specifies the parent project name of a PLC application. After specifying the project name, the system configuration and control program can be saved under this name. All data, which results during the creation of the configuration and the program, belongs to this parent project for this special automation. General identification for the complete set of programming and configuring information in the Project data bank, which displays the source code that describes the automation of a system.
<b>Project data bank</b>	The data bank in the Programming device, which contains the projection information for a Project.

---

**Prototype data file (Concept EFB)** The prototype data file contains all prototypes of the assigned functions. Further, if available, a type definition of the internal

---

**R**

**REAL** REAL stands for the data type "real". The input appears as Real literal or as Real literal with exponent. The length of the data element is 32 bit. The value range for variables of this data type reaches from 8.43E-37 to 3.36E+38.

**Note:** Depending on the mathematic processor type of the CPU, various areas within this valid value range cannot be represented. This is valid for values nearing ZERO and for values nearing INFINITY. In these cases, a number value is not shown in animation, instead NAN (**Not A Number**) oder INF (**INFinite**).

**Real literal** Real literals function as the input of real values in the decimal system. Real literals are denoted by the input of the decimal point. The values may be preceded by the signs (+/-). Single underline signs ( \_ ) between figures are not significant.

Example

-12.0, 0.0, +0.456, 3.14159\_26

**Real literal with exponent** Real literals with exponent function as the input of real values in the decimal system. Real literals with exponent are denoted by the input of the decimal point. The exponent sets the key potency, by which the preceding number is multiplied to get to the value to be displayed. The basis may be preceded by a negative sign (-). The exponent may be preceded by a positive or negative sign (+/-). Single underline signs ( \_ ) between figures are not significant. (Only between numbers, not before or after the decimal point and not before or after "E", "E+" or "E-")

Example

-1.34E-12 or -1.34e-12

1.0E+6 or 1.0e+6

1.234E6 or 1.234e6

**Reference** Each direct address is a reference, which starts with an ID, specifying whether it concerns an input or an output and whether it concerns a bit or a word. References, which start with the code 6, display the register in the extended memory of the state RAM.

0x area = Discrete outputs  
1x area = Input bits  
3x area = Input words  
4x area = Output bits/Marker words  
6x area = Register in the extended memory

**Note:** The x, which comes after the first figure of each reference type, represents a five figure storage location in the application data store, i.e. if the reference 400201 signifies a 16 bit output or marker word in the address 201 of the State RAM.

**Register in the extended memory (6x reference)** 6x references are marker words in the extended memory of the PLC. Only LL984 user programs and CPU 213 04 or CPU 424 02 can be used.

**RIO (Remote I/O)** Remote I/O provides a physical location of the I/O coordinate setting device in relation to the processor to be controlled. Remote inputs/outputs are connected to the consumer control via a wired communication cable.

**RP (PROFIBUS)** RP = Remote Peripheral

**RTU mode** Remote Terminal Unit  
The RTU mode is used for communication between the PLC and an IBM compatible personal computer. RTU works with 8 data bits.

**Rum-time error** Error, which occurs during program processing on the PLC, with SFC objects (i.e. steps) or FFBS. These are, for example, over-runs of value ranges with figures, or time errors with steps.

---

**S**

<b>SA85 module</b>	The SA85 module is a Modbus Plus adapter for an IBM-AT or compatible computer.
<b>Section</b>	<p>A section can be used, for example, to describe the functioning method of a technological unit, such as a motor.</p> <p>A Program or DFB consist of one or more sections. Sections can be programmed with the IEC programming languages FBD and SFC. Only one of the named programming languages can be used within a section.</p> <p>Each section has its own Document window in Concept. For reasons of clarity, it is recommended to subdivide a very large section into several small ones. The scroll bar serves to assist scrolling in a section.</p>
<b>Separator format (4:00001)</b>	The first figure (the Reference) is separated from the ensuing five figure address by a colon (:).
<b>Sequence language (SFC)</b>	The SFC Language elements enable the subdivision of a PLC program organizational unit in a number of Steps and Transitions, which are connected horizontally by aligned Connections. A number of actions belong to each step, and a transition condition is linked to a transition.
<b>Serial ports</b>	With serial ports (COM) the information is transferred bit by bit.
<b>Source code data file (Concept EFB)</b>	The source code data file is a usual C++ source file. After execution of the menu command <b>Library</b> → <b>Generate data files</b> this file contains an EFB code framework, in which a specific code must be entered for the selected EFB. To do this, click on the menu command <b>Objects</b> → <b>Source</b> .
<b>Standard format (400001)</b>	The five figure address is located directly after the first figure (the reference).
<b>Standardized literals</b>	<p>If the data type for the literal is to be automatically determined, use the following construction: 'Data type name'#'Literal value'.</p> <p>Example</p> <p>INT#15 (Data type: Integer, value: 15),</p> <p>BYTE#00001111 (data type: Byte, value: 00001111)</p> <p>REAL#23.0 (Data type: Real, value: 23.0)</p> <p>For the assignment of REAL data types, there is also the possibility to enter the value in the following way: 23.0.</p> <p>Entering a comma will automatically assign the data type REAL.</p>

<b>State RAM</b>	The state RAM is the storage for all sizes, which are addressed in the user program via References (Direct display). For example, input bits, discrettes, input words, and discrete words are located in the state RAM.
<b>Statement (ST)</b>	Instructions are "commands" of the ST programming language. Instructions must be terminated with semicolons. Several instructions (separated by semi-colons) can occupy the same line.
<b>Status bits</b>	There is a status bit for every node with a global input or specific input/output of Peer Cop data. If a defined group of data was successfully transferred within the set time out, the corresponding status bit is set to 1. Alternatively, this bit is set to 0 and all data belonging to this group (of 0) is deleted.
<b>Step</b>	SFC Language element: Situations, in which the Program behavior follows in relation to the inputs and outputs of the same operations, which are defined by the associated actions of the step.
<b>Step name</b>	<p>The step name functions as the unique flag of a step in a Program organization unit. The step name is automatically generated, but can be edited. The step name must be unique throughout the whole program organization unit, otherwise an Error message appears.</p> <p>The automatically generated step name always has the structure: S_n_m</p> <p>S = Step n = Section number (number running) m = Number of steps in the section (number running)</p>
<b>Structured text (ST)</b>	ST is a text language according to IEC 1131, in which operations, e.g. call up of Function blocks and Functions, conditional execution of instructions, repetition of instructions etc. are displayed through instructions.
<b>Structured variables</b>	<p>Variables, one of which is assigned a Derived data type defined with STRUCT (structure).</p> <p>A structure is a collection of data elements with generally differing data types (Elementary data types and/or derived data types).</p>
<b>SY/MAX</b>	In Quantum control devices, Concept closes the mounting on the I/O population SY/MAX I/O modules for RIO control via the Quantum PLC with on. The SY/MAX remote subrack has a remote I/O adapter in slot 1, which communicates via a Modicon S908 R I/O system. The SY/MAX I/O modules are performed when highlighting and including in the I/O population of the Concept configuration.
<b>Symbol (Icon)</b>	Graphic display of various objects in Windows, e.g. drives, user programs and Document windows.

---

**T**

<b>Template data file (Concept EFB)</b>	The template data file is an ASCII data file with a layout information for the Concept FBD editor, and the parameters for code generation.
<b>TIME</b>	TIME stands for the data type "Time span". The input appears as Time span literal. The length of the data element is 32 bit. The value range for variables of this type stretches from 0 to $2^{\text{exp}(32)}-1$ . The unit for the data type TIME is 1 ms.
<b>Time span literals</b>	<p>Permitted units for time spans (TIME) are days (D), hours (H), minutes (M), seconds (S) and milliseconds (MS) or a combination thereof. The time span must be denoted by the prefix t#, T#, time# or TIME#. An "overrun" of the highest ranking unit is permitted, i.e. the input T#25H15M is permitted.</p> <p>Example t#14MS, T#14.7S, time#18M, TIME#19.9H, t#20.4D, T#25H15M, time#5D14H12M18S3.5MS</p>
<b>Token</b>	The network "Token" controls the temporary property of the transfer rights via a single node. The token runs through the node in a circulating (rising) address sequence. All nodes track the Token run through and can contain all possible data sent with it.
<b>Traffic Cop</b>	The Traffic Cop is a component list, which is compiled from the user component list. The Traffic Cop is managed in the PLC and in addition contains the user component list e.g. Status information of the I/O stations and modules.
<b>Transition</b>	The condition with which the control of one or more Previous steps transfers to one or more ensuing steps along a directional Link.

---

**U**

<b>UDEFB</b>	User defined elementary functions/function blocks Functions or Function blocks, which were created in the programming language C, and are available in Concept Libraries.
<b>UDINT</b>	UDINT stands for the data type "unsigned double integer". The input appears as Integer literal, Base 2 literal, Base 8 literal or Base 16 literal. The length of the data element is 32 bit. The value range for variables of this type stretches from 0 to $2^{exp(32)}-1$ .
<b>UINT</b>	UINT stands for the data type "unsigned integer". The input appears as Integer literal, Base 2 literal, Base 8 literal or Base 16 literal. The length of the data element is 16 bit. The value range for variables of this type stretches from 0 to $(2^{exp16})-1$ .
<b>Unlocated variable</b>	<p>Unlocated variables are not assigned any state RAM addresses. They therefore do not occupy any state RAM addresses. The value of these variables is saved in the system and can be altered with the reference data editor. These variables are only addressed by symbolic names.</p> <p>Signals requiring no peripheral access, e.g. intermediate results, system tags etc, should primarily be declared as unlocated variables.</p>

---

**V**

<b>Variables</b>	<p>Variables function as a data exchange within sections between several sections and between the Program and the PLC. Variables consist of at least a variable name and a Data type. Should a variable be assigned a direct Address (Reference), it is referred to as a Located variable. Should a variable not be assigned a direct address, it is referred to as an unlocated variable. If the variable is assigned a Derived data type, it is referred to as a Multi-element variable. Otherwise there are Constants and Literals.</p>
<b>Vertical format</b>	Vertical format means that the page is higher than it is wide when looking at the printed text.

---

**W**

- Warning** When processing a FFB or a Step a critical status is detected (e.g. critical input value or a time out), a warning appears, which can be viewed with the menu command **Online** → **Event viewer...** . With FFBs the ENO output remains at "1".
- WORD** WORD stands for the data type "Bit sequence 16". The input appears as Base 2 literal, Base 8 literal or Base 16 literal. The length of the data element is 16 bit. A numerical range of values cannot be assigned to this data type.
-



---

## Index

---



### A

#### Arithmetic

- AVE\_\*\*\*, 19
- NEG\_\*\*\*, 87
- SIGN\_\*\*\*, 95

AVE\_\*\*\*, 19  
Averaging, 19  
AVGMV, 23  
AVGMV\_K, 27

### B

BCD\_TO\_INT, 31  
BCDConverter

- BCD\_TO\_INT, 31
- DBCD\_TO\_DINT, 53
- DBCD\_TO\_INT, 55
- DINT\_TO\_DBCD, 63
- INT\_TO\_BCD, 75
- INT\_TO\_DBCD, 77

BIT\_TO\_BYTE, 33  
BIT\_TO\_WORD, 37  
BYTE\_AS\_WORD, 41  
BYTE\_TO\_BIT, 43

### C

Conversion from 16 Bit BCD to INT, 31  
Conversion from 32 Bit BCD to DINT, 53  
Conversion from 32 Bit BCD to INT, 55  
Conversion from DINT to 32 Bit BCD, 63  
Conversion from INT to 16 Bit BCD, 75

Conversion from INT to 32 Bit BCD, 77  
Converter

- BIT\_TO\_BYTE, 33
- BIT\_TO\_WORD, 37
- BYTE\_AS\_WORD, 41
- BYTE\_TO\_BIT, 43
- DINT\_AS\_WORD, 61
- REAL\_AS\_WORD, 91
- TIME\_AS\_WORD, 99
- UDINT\_AS\_WORD, 111
- WORD\_AS\_BYTE, 113
- WORD\_AS\_DINT, 115
- WORD\_AS\_REAL, 117
- WORD\_AS\_TIME, 119
- WORD\_AS\_UDINT, 121
- WORD\_TO\_BIT, 123

#### Counter

- CTD\_\*\*\*, 45
- CTU\_\*\*\*, 47
- CTUD\_\*\*\*, 49
- DIVMOD\_\*\*\*, 65

CTD\_\*\*\*, 45  
CTU\_\*\*\*, 47  
CTUD\_\*\*\*, 49

### D

DBCD\_TO\_DINT, 53  
DBCD\_TO\_INT, 55  
Dead zone, 57  
DEAD\_ZONE\_REAL, 57  
Detecting and holding with rising edge, 93

Detection of all types of edges, 109  
DINT\_AS\_WORD, 61  
DINT\_TO\_DBCD, 63  
Division and Modulo, 65  
DIVMOD\_\*\*\*, 65  
Down counter, 45

## E

Edge detection  
  TRIGGER, 109  
EXTENDED  
  TOF\_P, 101  
  TON\_P, 105  
Extended  
  AVE\_\*\*\*, 19  
  AVGMV, 23  
  AVGMV\_K, 27  
  BCD\_TO\_INT, 31  
  BIT\_TO\_BYTE, 33  
  BIT\_TO\_WORD, 37  
  BYTE\_AS\_WORD, 41  
  BYTE\_TO\_BIT, 43  
  CTD\_\*\*\*, 45  
  CTU\_\*\*\*, 47  
  CTUD\_\*\*\*, 49  
  DBCD\_TO\_DINT, 53  
  DBCD\_TO\_INT, 55  
  DEAD\_ZONE\_REAL, 57  
  DINT\_AS\_WORD, 61

DINT\_TO\_DBCD, 63  
DIVMOD\_\*\*\*, 65  
HYST\_\*\*\*, 67  
INDLIM\_\*\*\*, 71  
INT\_TO\_BCD, 75  
INT\_TO\_DBCD, 77  
LIMIT\_IND\_\*\*\*, 79  
LOOKUP\_TABLE1, 83  
NEG\_\*\*\*, 87  
REAL\_AS\_WORD, 91  
SAH, 93  
SIGN\_\*\*\*, 95  
TIME\_AS\_WORD, 99  
TRIGGER, 109  
UDINT\_AS\_WORD, 111  
WORD\_AS\_BYTE, 113  
WORD\_AS\_DINT, 115  
WORD\_AS\_REAL, 117  
WORD\_AS\_TIME, 119  
WORD\_AS\_UDINT, 121  
WORD\_TO\_BIT, 123

## F

Floating mean with fixed window size, 23  
Floating mean with frozen correction factor,  
27  
Function  
  Parameterization, 11  
Function block  
  Parameterization, 11

## H

HYST\_\*\*\*, 67

## I

Indicator signal for delimiters with hysteresis,  
71  
Indicator signal for maximum value delimiter  
with hysteresis, 67  
INDLIM\_\*\*\*, 71  
INT\_TO\_BCD, 75  
INT\_TO\_DBCD, 77

**L**

Limit with indicator, 79  
LIMIT\_IND\_\*\*\*, 79  
LOOKUP\_TABLE1, 83

**M**

Measurement  
  AVGMV, 23  
  AVGMV\_K, 27  
  DEAD\_ZONE\_REAL, 57  
  HYST\_\*\*\*, 67  
  INDLIM\_\*\*\*, 71  
  LOOKUP\_TABLE1, 83  
  SAH, 93

**N**

NEG\_\*\*\*, 87  
Negation, 87

**O**

Off Delay with Pause, 101  
On Delay with Pause, 105

**P**

Parameterization, 11

**R**

REAL\_AS\_WORD, 91

**S**

SAH, 93  
Selection  
  LIMIT\_IND\_\*\*\*, 79  
Sign evaluation, 95  
SIGN\_\*\*\*, 95

**T**

TIME\_AS\_WORD, 99  
Timer  
  TOF\_P, 101  
  TON\_P, 105  
TOF\_P, 101  
TON\_P, 105  
Traverse progression with 1st degree  
interpolation, 83  
TRIGGER, 109  
Type conversion, 33, 37, 41, 43, 61, 91, 99,  
111, 113, 115, 117, 119, 121, 123

**U**

UDINT\_AS\_WORD, 111  
Up counter, 47  
Up/down counter, 49

**W**

WORD\_AS\_BYTE, 113  
WORD\_AS\_DINT, 115  
WORD\_AS\_REAL, 117  
WORD\_AS\_TIME, 119  
WORD\_AS\_UDINT, 121  
WORD\_TO\_BIT, 123

