

Concept  
IEC block library  
Part: LIB984

840 USE 504 00 eng Version 2.6



© 2002 Schneider Electric All Rights Reserved

---

---

## Table of Contents



---

<b>About the book</b> .....	<b>5</b>
<b>Part I General information on the LIB984 block library</b> .....	<b>7</b>
Overview .....	7
<b>Chapter 1 Parameterizing functions and function blocks</b> .....	<b>9</b>
Parameterizing functions and function blocks .....	9
<b>Chapter 2 At a glance</b> .....	<b>13</b>
Overview .....	13
Modsoft Functions and using the state RAM. ....	15
Concept uses variables with pre-defined, standardized types .....	15
Concept EFBs and Parameters. ....	16
Tables under Concept. ....	16
Concept EFBs and the ANY data type .....	17
Implementation aspects .....	17
<b>Part II EFB descriptions</b> .....	<b>19</b>
Overview .....	19
<b>Chapter 3 DLOG: Data event logging for PCMCIA Read/write support</b> ..	<b>21</b>
<b>Chapter 4 FIFO: First In/First Out stack register</b> .....	<b>27</b>
<b>Chapter 5 GET_3X: Reading 3x register</b> .....	<b>31</b>
<b>Chapter 6 GET_4X: Reading 4x register</b> .....	<b>33</b>
<b>Chapter 7 GET_BIT: Reading bit.</b> .....	<b>35</b>
<b>Chapter 8 IEC_BMDI: Block move</b> .....	<b>37</b>
<b>Chapter 9 LIFO: Last In/First Out stack register</b> .....	<b>45</b>
<b>Chapter 10 PUT_4X: Write 4x register</b> .....	<b>49</b>
<b>Chapter 11 R2T_***: Register to table</b> .....	<b>51</b>

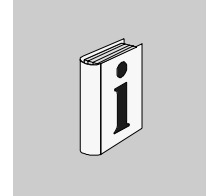
---

---

<b>Chapter 12</b>	<b>SET_BIT: Set bit</b> .....	<b>55</b>
<b>Chapter 13</b>	<b>SET_BITX: Set expanded bit</b> .....	<b>59</b>
<b>Chapter 14</b>	<b>SRCH_***: Search</b> .....	<b>63</b>
<b>Chapter 15</b>	<b>T2T: Table to table</b> .....	<b>67</b>
<b>Glossary</b>	.....	<b>73</b>
<b>Index</b>	.....	<b>97</b>

---

## About the book



---

### At a Glance

**Document Scope** This documentation should help you to configure functions and function blocks.

**Validity Note** This documentation is valid for Concept 2.6 under Microsoft Windows 98, Microsoft Windows 2000 and Microsoft Windows NT 4.x.

**Note:** Additional up-to-date tips can be found in the Concept README file.

---

### 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

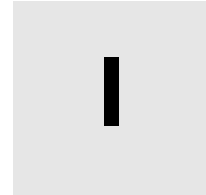
**User Comments** We welcome your comments about this document. You can reach us by e-mail at [TECHCOMM@modicon.com](mailto:TECHCOMM@modicon.com)

---



---

## General information on the LIB984 block library



---

### Overview

#### Introduction

This section contains general information about the LIB984 block library.

#### What's in this part?

This part contains the following chapters:

Chapter	Chaptername	Page
1	Parameterizing functions and function blocks	9
2	At a glance	13

General information

---



---

## **Parameterizing functions and function blocks**



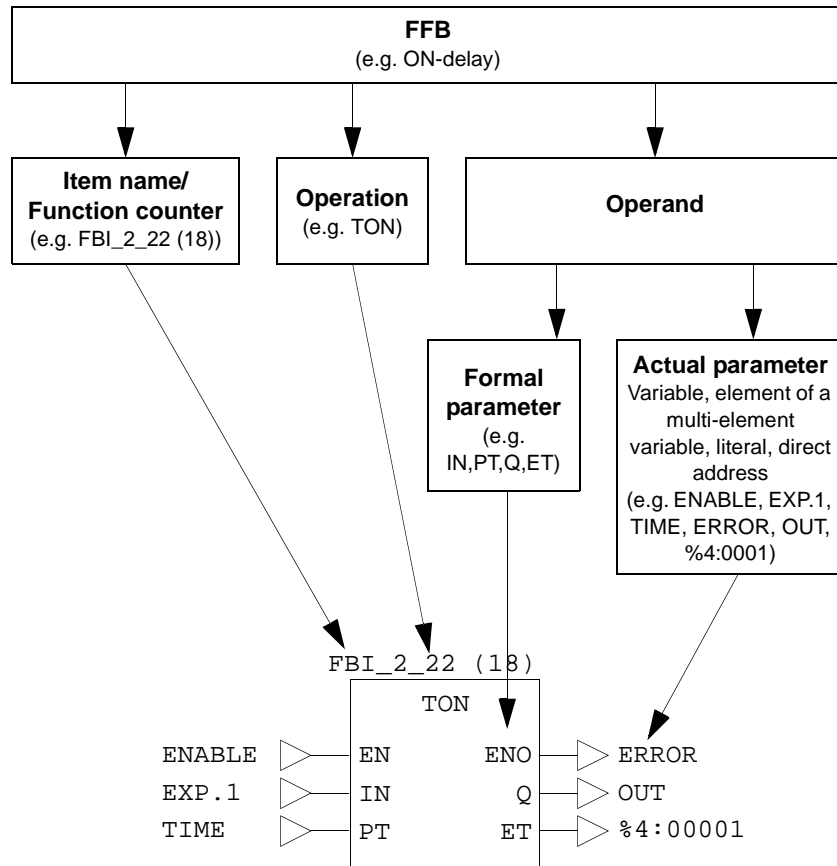
---

### **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.

---



---

## At a glance



---

### Overview

#### At a glance

The EFB Library LIB984 emulates the Modsoft functions in Concept without major differences. On the other hand all the features of Concept, different data types, located and unlocated variables, are available in these functions. Overview of the function blocks present:

Function block	Modsoft Equivalent
R2T_INT, R2T_UINT, R2T_DINT, R2T_UDINT, R2T_REAL	R->T
T2T	T->T
FIFO	FIN, FOUT
LIFO	
SRCH_INT, SRCH_UINT, SRCH_DINT, SRCH_UDINT, SRCH_REAL	SRCH
GET_3X	
GET_4X	
PUT_4X	
GET_BIT	
SET_BIT	

---

At a glance

---

**What's in this chapter?**

This chapter contains the following topics:

<b>Topic</b>	<b>Page</b>
Modsoft Functions and using the state RAM	15
Concept uses variables with pre-defined, standardized types	15
Concept EFBs and Parameters	16
Tables under Concept	16
Concept EFBs and the ANY data type	17
Implementation aspects	17

---

---

## Modsoft Functions and using the state RAM

---

<b>Introduction</b>	If we look at the details of the general implementation, we see that Modsoft functions only work with the State-RAM (0x, 1x, 3x and 4x registers) and process mainly 16-bit register values. All blocks or tables are based on 16-bit values.
<b>Conversion in Concept</b>	The size of these tables is given as the number of "words" required. For the different register types (0x, 1x, 3x, 4x) there is a physical address in the state RAM and they have a configured but fixed length. The length of each table is therefore known. All discreties are packed into words (16 bit = 1 word). Modsoft and Concept work directly on these packed bit structures. If bits are moved with the functions below then only structures of contiguous words are moved Example: Start addresses 0x1, 0x17,0xn with $n = i*16+1$
<b>Properties of Modsoft functions</b>	Modsoft functions have a maximum of 3 Boolean inputs, 3 Boolean outputs and a maximum of 3 parameters/parameter blocks and all inputs, outputs and parameters have no variable name.

---

## Concept uses variables with pre-defined, standardized types

---

<b>Differences between Concept and Modsoft</b>	<p>If we take a more detailed look at Concept, we notice the following differences:</p> <ul style="list-style-type: none"> <li>• Concept also has unlocated variables (no specific address), which may be randomly spread throughout memory. Addresses of variables may also change during runtime, when a Download Changes is invoked.</li> <li>• Concept also has a wide range of different data types of elementary types (INT, DINT, REAL) and defined structures to user-defined structures and arrays. EFBs and user-defined DFBs work with variables of these data types. EFBs can work with variables which are located to the State-Ram or with unlocated variables which may lie somewhere in the application RAM of the controller.</li> <li>• Blocks in Concept are pre-defined data-structures or arrays.</li> <li>• Tables in Concept are pre-defined arrays of elementary data types or structures.</li> </ul>
--	---

---

## Concept EFBs and Parameters

---

### Mode of functioning of Concept

The Concept compiler checks that each parameter variable has the same data type as the EFB pin. This is why several different ADD functions are necessary to cover for all the known elementary datatypes (integer, double integer or float, etc.). When defining an EFB for Concept each parameter must have a pre-defined datatype. If an EFB works with a special data-structure, then this data structure must be defined in Concept when implementing the EFB. During application program design it is not possible to connect a variable of another datatype to a certain pin of the EFB.

---

## Tables under Concept

---

### Mode of functioning of Concept

If we now look to "tables" then we see there are some difficulties. If an EFB works with a table, then this table must have been previously defined. If the table is an array of 10 integers this must be defined in the DTY data of the EFB library in the following way.

```
TYPE TI_10 :  
    ARRAY[0..9] OF INT;  
END_TYPE
```

When we now design an EFB using this array we can, for example, put/ get array-elements to/from the array. The EFB however only works with an array of 10 integers. If we need more elements, then we have to define a new (different) datatype. Even so the existing EFB will not work with this new data type as it is not the same as that defined during the design of the EFB.

The first problem is to define the right number of ARRAY elements.

The next problem is that we have to define different EFBs for arrays of different datatypes.

---



---

## Concept EFBs and the ANY data type

---

**Functioning mode of Concept** Concept has the special data type 'ANY' available for simplifying table operations. A pin declared with this data type can be linked to a variable of any data type. In order to work properly, at run time, the size (in bytes) of the data type of the connected variables is transferred to the EFB as a hidden parameter. However the actual type of the variable (whether it is an INT, UINT, WORD, a structure of elements or an array) is still unknown to the EFB.

---

**Example** The EFB is passed a pointer to a variable which has a size of 4 bytes. Whether these 4 bytes are of type DINT or REAL is unknown. For this reason the parameter ANY can only be used if the processing of this parameter is clearly defined. In the above example the EFB doesn't know whether to perform floating point or integer arithmetic. Should the size increase, it becomes more and more unclear as to what data type the variable is. (e.g. a size of 100 bytes could be an array of bytes, integers, or reals, or a structure with different data types)

---

**Further restrictions** Another restriction of the datatype ANY is, that if this type is used more than once in the same EFB, all pins of this type must be connected to variables of the same datatype.

---

## Implementation aspects

---

**Procedure** These differences between Concept and Modsoft outline what is possible for the conversion of the Modsoft functions to a Concept EFB library. In the implementation we have contrasting requirements:

- Implement the function conforming as much as possible to the existing solution in Modsoft.  
i.e. only implementing a state RAM solutions (cannot be used with localized variables)
- Implementing the function according to the standard programming rules for IEC1131.  
i.e. different solutions for parameters and consequently in mode of operation.

---

**Limitations** The Modsoft functions BLKM, BLKT and TBLK are not included in this package. BLKM already has an equivalent Concept function named MOVE. BLKT and TBLK can be replaced with a similar DFB function.

---

At a glance

---

---

## EFB descriptions



---

### Overview

#### Introduction

These EFB descriptions are listed in alphabetical order.

#### What's in this part?

This part contains the following chapters:

Chapter	Chaptername	Page
3	DLOG: Data event logging for PCMCIA Read/write support	21
4	FIFO: First In/First Out stack register	27
5	GET_3X: Reading 3x register	31
6	GET_4X: Reading 4x register	33
7	GET_BIT: Reading bit	35
8	IEC_BMDI: Block move	37
9	LIFO: Last In/First Out stack register	45
10	PUT_4X: Write 4x register	49
11	R2T_***: Register to table	51
12	SET_BIT: Set bit	55
13	SET_BITX: Set expanded bit	59
14	SRCH_***: Search	63
15	T2T: Table to table	67

EFB descriptions

---

---

## DLOG: Data event logging for PCMCIA Read/write support

3

---

### Overview

#### At a glance

This chapter describes the DLOG block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	22
Representation	23
Detailed description	24
Run time error	26

## Brief description

---

### Function description

The read/write support in PCMCIA consists of a configuration expansion established with a DLOG block. With the DLOG block, one application can copy data e.g. onto or from a PCMCIA flashcard, delete individual memory blocks on a PCMCIA flashcard, and delete an entire PCMCIA flashcard. The application regulates the data format and how often the data are saved.

As additional parameters, EN and ENO can be configured.

---

### Limitations

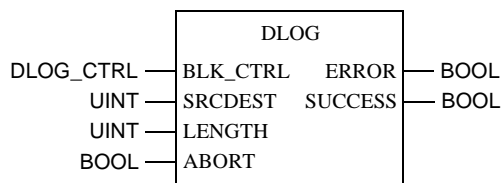
When using the DLOG block, the following limitations apply:

- This block is only available with the PLC family TSX Compact.
  - The DLOG block can only work with linear PCMCIA flash cards which use AMD flash devices.
-

## Representation

### Symbol

Representation of the function block:



### Description of parameters

Description of block parameters:

Parameters	Data type	Meaning
BLK_CTRL	DLOG_CTRL	Data structure of the Control block
SRCDEST	UINT	First 4x register in a data area, used as source or target for the operation specified.
LENGTH	UINT	Maximum number of registers reserved for the data area: 0 to 100.
ABORT	BOOL	1 = stops the current active operation
ERROR	BOOL	1 = Error (See also <i>Error message, p. 26</i> ) during DLOG operation (operation terminated unsuccessfully)
SUCCESS	BOOL	1 = DLOG operation terminated successfully (operation successful)

### DLOG\_CTRL data structure

Description of the DLOG\_CTRL data structure:

Element	Data type	Meaning
status	WORD	Error status
operation	WORD	Operation type
window	WORD	Window (block indicator)
offset	WORD	Offset (Byte address within the block)
count	WORD	counter

**DLOG\_CTRL**

**Detailed description**

The structured variable entered at this input contains the five elements for the DLOG control block. The control block is used to define the function of the DLOG command, the PCMCIA flash card window and the PCMCIA flashcard window offset, a returned status word and a data word counter value.

The control block is used to define the function of the DLOG command, the PCMCIA flash card window and the PCMCIA flashcard window offset, a returned status word and a data word counter value.

Element	Meaning	Function
status	Error status	Displays DLOG errors in HEX values: <ul style="list-style-type: none"> <li>● 1 = the counter parameter of the control block, i.e. the DLOG block length during a write operation (operation type = 1)</li> <li>● 2 = PCMCIA card command failed during startup (write/read/delete)</li> <li>● 3 = PCMCIA card command failed during execution (write/read/delete)</li> <li>● 4 = current EXEC (16-bit) is not a valid EXEC for Compact (32-bit).</li> </ul>
operation	Operation type	The following operations are available: <ul style="list-style-type: none"> <li>● 1 = write to PCMCIA card</li> <li>● 2 = read to PCMCIA card</li> <li>● 3 = delete one block</li> <li>● 4 = delete entire card contents</li> </ul>
window	Window (block indicator)	This element designates a specific block (PCMCIA save window) on the PCMCIA card (1 block = 128 Kbytes). The number of blocks depends on the PCMCIA card memory size. (e.g. 0 ... Max. 31 for a 4 meg (PCMCIA card.)
offset	Offset (Byte address within the block)	Specific byte area within a specific block on the PCMCIA card. Area: 1 ... 128 Kbyte
count	counter	Number of 4x registers written or read onto the PCMCIA card. Area: 0 to 100.

**Note:** PCMCIA flash card addresses are addresses with a window offset basis. Established size of windows is 128 Kbyte (65,535 words (16-bit values)). No write/read operation can overshoot the boundary between one window and the next. for this reason "offset" plus "count" must always be  $\leq$  128 Kbyte (65.535 words).



**SRCDEST** The value entered at this input defines the first register (e.g. the value "50" produces the 4x register address 4x000050) in an associated block of 4x word registers. The DLOG block will use this block as the source or destination of the operation established in the "operation" of the BLK\_CTRL input.

Table of operations:

Operation	State RAM reference	Function
Write	4x	Source reference
Read	4x	Destination reference
Delete block	none	none
Delete card	none	none

If the value at this input is not within the 4x register area, an error message appears and the ERROR output is set to "1".

**LENGTH** The value entered at this input is the length of the data range – i.e. the maximum number of words (registers) authorized in a transfer to or from the PCMCIA flash card. The length can be in the 0 to 100 range.

**ABORT** If there is a "1" at this input, the current operation is aborted. The input is static, i.e. as long as the value is "1", the block is not executed.

**ERROR** If the current DLOG operation has been terminated unsuccessfully, this output is set to "1".

A read or write operation can take on several cycles before terminating. This means that if ERROR and SUCCESS both have the value "0", the current operation has not yet been completed.

**SUCCESS** If the current DLOG operation has been completed successfully, this output is set to "1".

A read or write operation can take on several cycles before terminating. This means that if ERROR and SUCCESS both have the value "0", the current operation has not yet been completed.

## Runtime error

---

### Error message

An error message appears if

- the current EXEC (16-bit) is not a valid EXEC for Compact (32-bit). In this case the "status" element of the BLK\_CTRL input is set to the value "4" (HEX) and the Error message "E\_EFB\_CURRENT\_MODE\_NOT\_ALLOWED" is generated. Otherwise the ERROR output is set to "1" and the execution of the block is then aborted. (If the block continues to be operated with the incompatible EXEC, the error message appears in each program cycle as long as the block is active.)
  - the value at the SRCDEST is not within the 4x register area. In this case an error message is generated. Otherwise the ERROR output is set to "1" and the execution of the block is then aborted.
  - an error occurs during a read or write cycles. In this case the PCM error code is displayed in the "status" element of the BLK\_CTRL input. Otherwise the ERROR output is set to "1" and the execution of the block is then aborted.
-

---

## FIFO: First In/First Out stack register



# 4

---

### Overview

#### At a glance

This chapter describes the FIFO 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

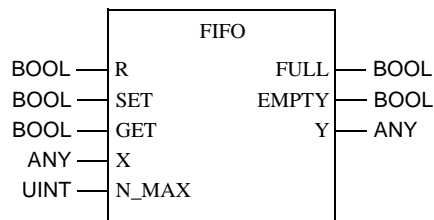
This Function block is a 'first in – first out' stack register. As additional parameters, EN and ENO can be configured.

**Note:** For technical reasons, this function block cannot be used with the programming languages ST and IL.

## Representation

### Symbol

Block representation:



### Parameter description

Description of the block parameters:

Parameter	Data type	Meaning
R	BOOL	1 = stack will be cleared
SET	BOOL	1 = write value to stack
GET	BOOL	1 = read value from stack
X	ANY should be a field (array) from ANY_ELEM e.g. ARRAY[0..X] OF INT	Stack input
N_MAX	UINT	Maximum number of elements in the stack
FULL	BOOL	1= stack is full, no further elements can be put in the stack.
EMPTY	BOOL	1 = stack is empty (number of elements in the stack = 0)
Y	ANY should be a field (array) from ANY_ELEM e.g. ARRAY[0..X] OF INT	Stack output

**Detailed description**

**Functionality Under Concept**

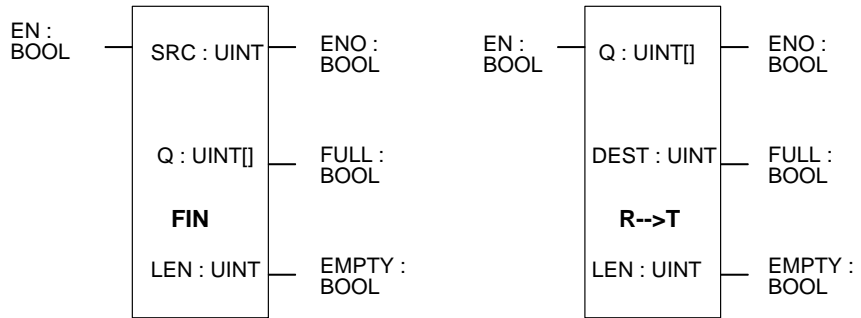
FIFO is a 'first in-first out' stack register  
 The index and stack registers are invisible to the user. The stack register is a component of the internal status and can take up to 2000 bytes of data (i.e. 1000 INT- or 500 REAL- or 500 TIME elements).  
 The function block contains two Boolean inputs GET and SET to read a value from the stack register or to write a value to the stack register. As long as this input is set to 1, a value is either read from or written to the FIFO stack. If GET and SET are set at the same time SET (write) is executed first and then GET (read). The stack register is cleared if R(eset)=1.  
 The input parameters that monitor the stack must be set in a sensible sequence so that the function block works properly.  
 A sensible sequence would be e.g.:

Cycle	Parameter	Result
Cycle n	R=0, SET=0, GET=0	Stack not initialized
Cycle n+1	R=1, SET=0, GET=0	Stack initialized
Cycle n+2	R=0, SET=0, GET=0	End initialization
Cycle n+3	R=0, SET=1, GET=0	Load stack with x values
Cycle n+x+1	R=0, SET=0, GET=0	End load
Cycle n+x+2	R=0, SET=0, GET=1	Get x values
Cycle n+x+2+x	R=0, SET=0, GET=1	Stack not empty

The N\_MAX parameter indicates the maximum number of elements in the stack register.  
 If the stack register is full (number of elements in the stack register=N\_MAX <= 2000 / (Size of (X)) then FULL is set to 1 for one scan and no other elements can be put on the stack register. If the stack register is empty (number of elements in the stack register = 0) then EMPTY is set to 1 for one scan. The function has an X input and a Y output for different elementary data types.  
 X and Y are of type ANY which implies a predefined length. Because of the limited size of the internal stack register, only input and output types are accepted with an elementary size equal to or less than 200 bytes. Otherwise, a runtime error occurs and an error message is generated which sets ENO to 0.

**Functionality under Modsoft**

These functions copy a value from a source register (16 bits) to a queue (table) or vice-versa.



The table begins with the first address of the queue registers. The first element in this table is the number of those elements defined in the queue. These functions copy a 16 bit value from a source register to the queue (Index + 1) in every cycle.

Entry, Input/Output	Meaning
Upper entry	Source is a reference to the State RAM: 0x, 1x or in a combined 16 bit field 3x, 4x
Middle entry	Queue 'Pointer' is a reference to the State RAM: Beginning of 4x register + 1 + value in pointer
Lower entry	Table length (1..100).
Upper input/upper output	Function enable
Middle output	Queue full, further copy procedures will not be executed
Lower output	Queue empty

**Differences**

Differences between Concept and Modsoft:

- The table can have any predefined data structure. Modsoft functions cannot be processed with localized variables or links. Modsoft uses offsets in the State RAM. The table is a component of the State RAM (not reserved, no check for multiple access).
- Pins are recognized under Concept.  
EN/ENO are optional under Concept (Standard IEC1131-3).
- Different function names ('->' cannot be used in one name under Concept).
- Different display (see *Functionality Under Modsoft*, p. 30).

---

## GET\_3X: Reading 3x register



---

### Overview

#### At a glance

This chapter describes the GET\_3X block

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	32
Representation	32
Detailed description	32

## Brief description

---

**Function description** This function block writes values from the 3x register region of the state RAM to the variable that is connected to the output pin. EN and ENO can be configured as additional parameters.

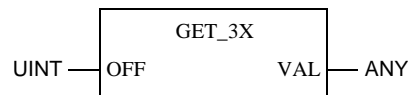
**Note:** For technical reasons, this function block can only be used in compact form in ST or IL programming languages (`INSTANCE_NAME (OFF:=offset, VAL=>value)`).

---

## Representation

---

**Symbol** Representation of the function block:



**Description of parameters** Description of the block parameters:

Parameters	Data type	Meaning
OFF	UINT	OFF is an offset in the 3x register memory.
VAL	ANY	Output

---

## Detailed description

---

**Function description** The function block GET\_3X writes values from the 3x register region of the state RAM into the variable, which is connected to the output pin. OFF is an offset in the 3x register memory. The function copies as many bytes as the size of the output datatype connected to the output pin.

**Example** The output of this function will read the 16-bit value of register 300120, if OFF = 120 and the output is of datatype INT. The value in OFF may be modified at runtime. If the value in OFF is beyond the configured number of 3x-Registers, an error message is generated and ENO is set to 0.

---



---

## GET\_4X: Reading 4x register



---

### Overview

#### At a glance

This chapter describes the GET\_4X block

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	34
Representation	34
Detailed description	34

## Brief description

---

**Function description** This function block writes values from the 4x register region of the state RAM to the variable that is connected to the output pin. EN and ENO can be configured as additional parameters.

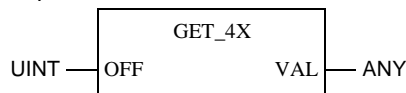
**Note:** For technical reasons, this function block can only be used in compact form in ST or IL programming languages (`INSTANCE_NAME (OFF:=offset, VAL=>value)`).

---

## Representation

---

**Symbol** Representation of the function block:



**Description of parameters** Description of the block parameters:

Parameters	Data type	Meaning
OFF	UINT	OFF is an offset in the 4x register memory.
VAL	ANY	Output

---

## Detailed description

---

**Function description** The function block GET\_4X writes values from the 4x register region of the state RAM into the variable, which is connected to the output pin. OFF is an offset in the 4x register memory. The function copies as many bytes as the size of the output datatype connected to the output pin.

---

**Example** The output of this function will read the 16-bit value of register 400120, if OFF = 120 and the output is of datatype INT. The value in OFF may be modified during runtime. If OFF lies outside the configured number of 4x-Registers, an error message is generated and ENO is set to 0.

---

---

## GET\_BIT: Reading bit



---

### Overview

#### At a glance

This chapter describes the GET\_BIT block

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	36
Representation	36
Detailed description	36

## Brief description

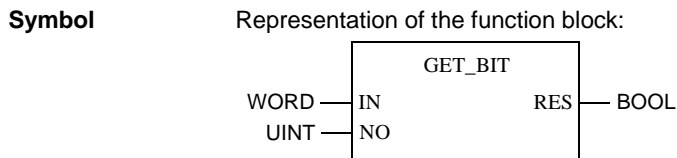
---

**Function description** This function block reads the bit of the "IN" input selected with "NO" and writes the current state to the "RES" output.  
As additional parameters, EN and ENO can be configured.

---

## Representation

---



**Description of parameters** Description of the block parameters:

Parameters	Data type	Meaning
IN	WORD	Input register
NO	UINT	Bit number to be read.
RES	BOOL	Current state of selected bit.

---

## Detailed description

---

**Function description** The GET\_BIT function block reads the bit of the "IN" input selected with "NO" and writes the current state to the "RES" output.  
The output is the current state of the selected input data bit.  
The "NO" parameter shows which input data bit to select.



---

## IEC\_BMDI: Block move



---

### Overview

#### At a glance

This chapter describes the IEC\_BMDI block

#### What's in this chapter?


This chapter contains the following topics:

Topic	Page
Brief description	38
Representation	38
Detailed description	40
Run time error	43

## Brief description

### Function description

This function block creates a word-by-word copy of the number of elements listed in LENGTH from the OFF\_IN position in the source table (SEL\_IN) to the OFF\_OUT position in the destination table (SEL\_OUT).

	<b>WARNING</b>
	<p><b>Dangerous process conditions</b></p> <p>This function block overwrites the values in State memory WITHOUT regard to possible forced values in the reference data editor. This can produce serious process conditions.</p> <p><b>Failure to observe this precaution can result in severe injury or equipment damage.</b></p>

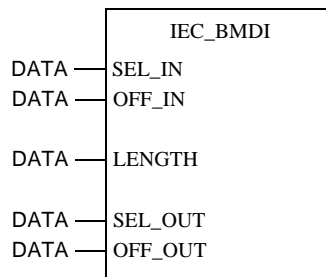
As additional parameters, EN and ENO can be configured.

**Note:** For technical reasons, this function block cannot be used with the programming language IL.

## Representation

### Symbol

Representation of the function block:




**Description of parameters**

Description of the block parameters:

Parameters	Data type	Meaning
SEL_IN	UINT	Source table having contents copied. Selection of source table: 0 = 0x 1 = 1x 3 = 3x 4 = 4x
OFF_IN	UINT	Offset in selected source table (SEL_IN). As the copy operation is executed word-by-word, OFF_IN must be a multiple of n+1 for 0x and 1x source tables (n=0, 1, 2, 3... e.g. 1, 17, 33, 49 etc.) The offset is subject to upper limit supervision and must be within the limits of the source table.
LENGTH	UINT	Table size for source and destination table. LENGTH indicates the number of elements in the source table to be copied. Because the copy operation is performed word-by-word, 0x and 1x source tables must have a multiple of 16 (e.g. 16, 32, 48, etc.) for LENGTH. LENGTH is subject to upper limit supervision and must be within the limits of the source and destination tables. Independent of the configured limits, the LENGTH value was additionally restricted to the following values to prevent the copy operation from taking up too much time: 0x, 1x bits: max LENGTH = 1600 3x, 4x registers: max LENGTH = 100
SEL_OUT	UINT	Destination table to which contents of source table will be copied. Selection of destination table: 0 = 0x 4 = 4x
OFF_OUT	UINT	Offset in selected source table. Because the copy operation is performed word-by-word, 0x and 1x source tables OFF_OUT must have a multiple of n+1 (n=0, 1, 2, 3... e.g. 1, 17, 33, 49, etc.). The offset is subject to upper limit supervision and must be within the limits of the destination table.

## Detailed description

### Function description

	<b>WARNING</b>
	<p><b>Dangerous process conditions</b></p> <p>This function block overwrites the values in State memory WITHOUT regard to possible forced values in the reference data editor. This can produce dangerous process conditions.</p> <p><b>Failure to observe this precaution can result in severe injury or equipment damage.</b></p>

IEC\_BMDI makes a word-by-word copy of the number of elements listed in LENGTH from the OFF\_IN position in the source table (SEL\_IN) to the OFF\_OUT position in the destination table (SEL\_OUT). While copying, LENGTH always uses the type of SEL\_IN for orientation (0x, 1x: LENGTH = number of bits; 3x, 4x: LENGTH = number of words)

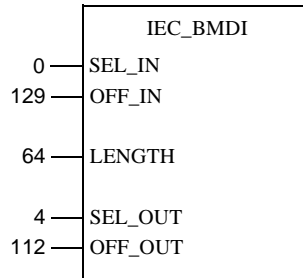
Copy behavior:

- 0x or 1x to 0x  
The source and target lengths are identical during copying of 0x or 1x to 0x
- 3x or 4x to 4x  
The source and target lengths are identical during copying of 3x or 4x to 3x
- 0x or 1x to 4x  
When copying 0x or 1x to 4x, a WORD-to-WORD copy is made as well. In this, the first source bit is copied into the MSB (highest bit) of the first 4x register, and so on. LENGTH defines the number of bits to be copied. The maximum output length is LENGTH/16 registers.
- 3x or 4x to 0x  
When copying 3x or 4x to 0x, a WORD-to-WORD copy is made as well. In this, the MSB (highest bit) of the first register is copied into the first destination bit and so on. LENGTH defines the number of registers to be copied. The maximum output length is LENGTH x 16 bits.

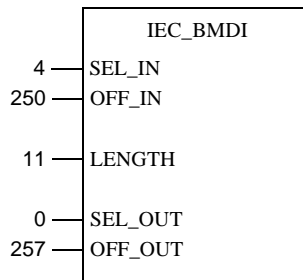


**Example 1**

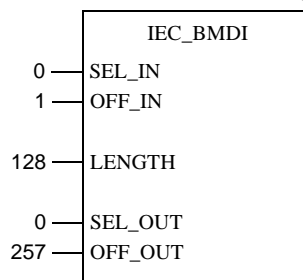
In the example, 64 0x source bits from start address 0:00129 are copied into the 4x destination register (starting at address 4:00112). The input range is 0:00129 to 0:00192, and the output range is 4:00112 to 4:00115.

**Example 2**

In the example, 11 4x source registers from start address 4:00250 are copied into the 0x destination bits (starting at address 0:00257). The input range is 4:00250 to 4:00260, and the output range is 0:00257 to 0:00432.

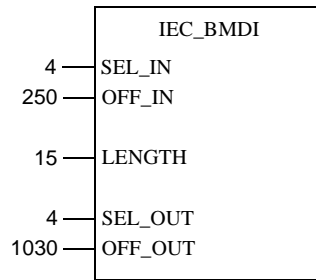
**Example 3**

In the example, 128 0x source bits are copied from start address 0:00001 into the 0x destination bits (starting at address 0:00257). The input range is 0:00001 to 0:00127, and the output range is 0:00257 to 0:00384.



**Example 4**

In the example, 15 4x source registers are copied from start address 4:00250 into the 4x destination registers (starting at address 4:01030). The input range is 4:00250 to 4:00264, and the output range is 4:01030 to 4:01044.



## Runtime error

### Runtime error

The following standard user error messages are utilized:

User error message	Meaning
E_EFB_USER_ERROR_1	Input value is invalid register type (SEL_IN).
E_EFB_USER_ERROR_2	The input offset (OFF_IN) selects an address outside acceptable limits.
E_EFB_USER_ERROR_3	The input offset (OFF_IN) is not 1 or a multiple of 16+1.
E_EFB_USER_ERROR_4	Output value is invalid register type (SEL_OUT).
E_EFB_USER_ERROR_5	The output offset (OFF_IN) selects an address outside acceptable limits.
E_EFB_USER_ERROR_6	The output offset (OFF_OUT) is not 1 or a multiple of 16+1.
E_EFB_USER_ERROR_7	The value for LENGTH is 0.
E_EFB_USER_ERROR_8	The value for LENGTH addresses more than 1600 bits.
E_EFB_USER_ERROR_9	The value for LENGTH addresses more than 100 words.
E_EFB_USER_ERROR_10	The value for LENGTH selects a source address outside the acceptable limits.
E_EFB_USER_ERROR_11	The value for LENGTH selects a destination address outside the acceptable limits.
E_EFB_USER_ERROR_12	The value for LENGTH is not a multiple of 16.
E_EFB_USER_ERROR_13	Warning: Address overlap of input and output addresses.

If there are no errors, the function block copies the values from the indicated source to the destination address and sets the ENO output to 1.

The user errors 1 through 12 will block the copy operation and are setting the ENO output to 0.

If user error 13 occurs, the copy operation continues and the ENO output remains at 1 because this error is treated as a warning. However, the user error will be reported in the online event dialog.

IEC\_BMDI: Block move

---

---

## LIFO: Last In/First Out stack register

9

---

### Overview

#### At a glance

This chapter describes the LIFO block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	46
Representation	46
Detailed description	47

---

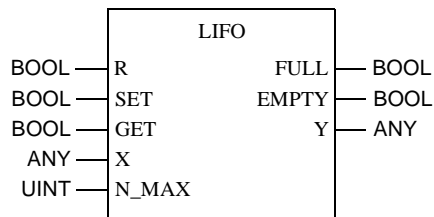
## Brief description

**Function description** This Function block is a 'last in – first out' stack register. As additional parameters, EN and ENO can be configured.

**Note:** For technical reasons, this function block cannot be used with the programming languages ST and IL.

## Representation

**Symbol** Representation of the function block:



**Description of parameters** Description of the block parameters:

Parameters	Data type	Meaning
R	BOOL	1 = stack register will be cleared
SET	BOOL	1 = write value into stack register
GET	BOOL	1 = read value from stack register
X	ANY should be Array ANY_ELEM e.g. .ARRAY[0..X] OF INT	Stack register input
N_MAX	UINT	Maximum number of elements in the stack register
FULL	BOOL	1= stack register is full, cannot put any more elements onto stack register.
EMPTY	BOOL	1 = stack register is empty (number of elements in the stack register = 0)
Y	ANY should be Array ANY_ELEM e.g. .ARRAY[0..X] OF INT	Stack register output

## Detailed description

### Function description

LIFO is a 'last in/first out' stack register.

The index and stack register are invisible to the user. The stack register is part of the internal state and can accept up to 2000 bytes (i.e. 1000 INT or 500 REAL or 500 TIME elements).

The function block has two Boolean inputs, GET and SET, that are used to either read a value from or write a value to the stack register. If GET and SET are set simultaneously, SET (write) will be executed before GET (read).

The stack register will be cleared if R(eset) = 1.

The input parameters checking the stack must be set in a meaningful order allow the function block to work properly.

A meaningful order is, for instance:

Cycle	Parameters	Result
Cycle n	R=0, SET=0, GET=0	Stack not initialized
Cycle n+1	R=1, SET=0, GET=0	Stack initialized
Cycle n+2	R=0, SET=0, GET=0	Ending initializing
Cycle n+3	R=0, SET=1, GET=0	Loading stack with x values
Cycle n+x+1	R=0, SET=0, GET=0	Ending loading
Cycle n+x+2	R=0, SET=0, GET=1	Get x values
Cycle n+x+2+x	R=0, SET=0, GET=1	Stack not empty

An N\_MAX parameter defines the maximum number of elements in the stack register.

In a full stack register (number of elements in the stack register = N\_MAX <= 2000 / (size of (X)), FULL is set to 1. Cannot put any more elements onto stack register. In an empty stack register (number of elements in the stack register = 0), EMPTY is set to 1.

The function has one X input and one Y output with various data types.

X and Y are type ANY which implicates a pre-defined length. Due to the limited size of the stack register, only data types with an element size smaller than or equal to 200 bytes are allowed. Otherwise a runtime error will be generated and the ENO output is set to 0.

LIFO: Last In/First Out stack register

---



---

## PUT\_4X: Write 4x register

10

---

### Overview

#### At a glance

This chapter describes the PUT\_4X block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief Description	50
Representation	50
Detailed description	50

## Brief Description

---

**Function Description** This function block reads values from the IN variables and writes them in the 4x register range of the State RAM.

**Note:** As this function block has no output pin, the editors do not recognize that this function block overwrites a 4x register range. For this reason these 4x registers in the **Used Reference Display** dialog box are not displayed as used.

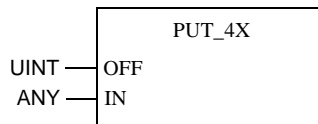
As additional parameters, EN and ENO can be configured.

---

## Representation

---

**Symbol** Representation of the function block:



**Description of parameters** Description of the block parameters:

Parameters	Data type	Meaning
OFF	UINT	OFF is an offset in the 4x register memory.
IN	ANY	Input

---

## Detailed description

---

**Function description** The function block PUT\_4X reads values from the variable IN and writes them into the 4x register region of state RAM.  
 OFF is an offset in the 4x register memory.  
 The function copies as many bytes as the size of the input datatype connected to the pin IN.

---

**Example** The function will copy the 16-bit field from IN to the register 400120, if OFF = 120 and the input is a WORD.  
 The value of OFF may be modified during runtime.  
 If OFF lies outside the configured number of 4x-Registers, an error message is generated and ENO is set to 0.

---

---

## R2T\_\*\*\*: Register to table

11

---

### Overview

#### At a glance

This chapter describes the R2T\_\*\*\* block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	52
Representation	52
Detailed description	53

---

## Brief description

### Function description

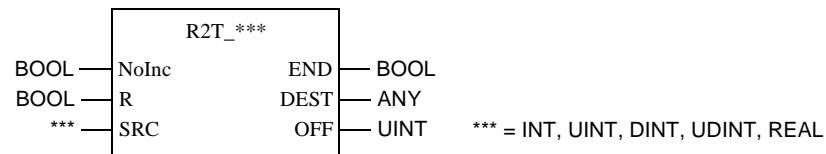
This function block copies the value entered in the SRC to the DEST parameter which is interpreted as a table. EN and ENO can be configured as additional parameters.

**Note:** Since this function block has an output of data type ANY but has no input of this data type the ANY output must be assigned with the => operator. An instance of this function block may be called one time only. Calling the same function block instance more than once is not possible.

## Representation

### Symbol

Representation of the function block:



### Description of parameters

Description of the block parameters:

Parameters	Data type	Meaning
NoInc	BOOL	1: freezes the pointer value
R	BOOL	1: resets the pointer value to zero
SRC	INT, UINT, DINT, UINT, REAL	Source data to be copied in the current cycle
END	BOOL	1: Pointer value = Table length, i.e. table is full, the function block performs no further copy functions, and OFF is no longer incremented. The function block (and consequently END as well) can be reset with R=1.
DEST	ANY should be Array of INT, UINT, DINT, UDINT or REAL e.g. ARRAY[0..X] OF INT	Destination register where source data will be copied in the cycle.
OFF	UINT	OFF shows the position in the table. OFF is normalized with a reset (R), i.e. when R=1, OFF is set to "1". After the restore, OFF is incremented by 1.

## Detailed description

### Mode of functioning under Concept

R2T copies the value entered at SRC to the DEST parameter which is interpreted as a table.

The OFF parameter, an offset, points to the position in the destination field (array) where the source value is to be saved.

In each cycle, the function copies the value from SRC to DEST[OFF] and increments the offset value by data type size in the table, i.e. OFF+1.

The offset value is automatically raised in each cycle as long as the NoInc parameter does not have the value 1. OFF is of the read/write type, equivalent to the VAR\_IN\_OUT IEC parameters

Association between OFF, NoInc and R:

OFF (previous cycle)	NoInc	R	OFF (current cycle)	Comment
n (any value)	0	1	2	The value for OFF is reset to 1 with R=1, and since NoInc=0, incremented by 1 already in the same cycle.
n (any value)	1	1	1	The value for OFF is reset to 1 with R=1, and since NoInc=1, it is not incremented.
n (any value)	1	0	n	If NoInc=1, the value for OFF is not incremented and the value from the previous cycle is maintained.
n (any value)	0	0	n+1	If R=0 and NoInc=0, the value from the previous cycle is incremented by 1.

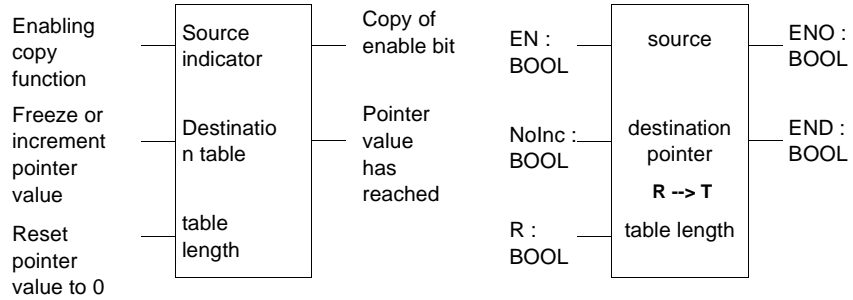
SRC has the same data type (INT, UINT, DINT, UDINT, REAL ) as the selected function block (refer to \_\*\*\*).

DEST is type ANY which implicates a pre-defined length. The same type is accepted as the data type for this Array as for SRC irrespective of the type entered for the run time (it could be a structure of various types).

OFF undergoes a bounds check in each cycle. If OFF exceeds the length of the table (internal parameter size), END is set to 1 and OFF is no longer increased (the function no longer copies until OFF accepts a value within the table's bounds again).

**Mode of functioning under Modsoft**

This function copies a value from a source register (16 bits) into a table.



The table starts with a destination register. The first element is the offset value in the table to which the source value has to be copied.

In each cycle, the function copies the 16-bit value from the source register into the destination register + offset + 1.

The offset is increased in each cycle when the mid input is 0.

The offset is reset to 0 when the lower input is 1.

Entry, Input/Output	Meaning
Upper entry	Source is a reference into State-Ram: 0x, 1x, or in a contiguous 16-bit field 3x, 4x
Mid entry	Destination 'pointer' is reference into State-Ram. Start of 4x register + 1 + value in pointer
Lower entry	Length of table (1..255, 1..999, depending on process)
Upper input, upper output	Enable function
Mid input	Raising offset for output table or not (0 or 1)
Lower input	Resetting offset for output table (if 1 then offset = 0)
Mid output	Offset reaching table size – no further copy operations

**Differences**

Differences between Concept and Modsoft:

- The table is any pre-defined data structure. Modsoft functions do not work with unlocated variables or links. Modsoft uses offsets in the state RAM. The table is part of the state RAM (not reserved, no checking for multiple use).
- Pins are named under Concept.
- EN/ENO are optional under Concept (Standard IEC1131-3).
- Various function names ('->' can not be used in a name under Concept).
- Various displays (see *Mode of functioning under Modsoft, p. 54*).

---

## SET\_BIT: Set bit

12

---

### Overview

#### At a glance

This chapter describes the SET\_BIT block.

#### What's in this chapter?

This chapter contains the following topics:


Topic	Page
Brief description	56
Representation	56
Detailed description	57

## Brief description

---

### Function description

This Function block sets the bit of the "RES" output selected with "NO" to the value of "IN".

	<b>WARNING</b>
	<p><b>Dangerous process conditions</b></p> <p>This function block overwrites the values in status RAM WITHOUT regard to possible forced values in the reference data editor. This can produce dangerous process conditions.</p> <p><b>Failure to observe this precaution can result in severe injury or equipment damage.</b></p>

The parameters EN and ENO can additionally be projected.

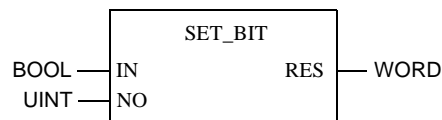
**Note:** For technical reasons, the function block cannot be used in programming languages ST or IL. If you wish to use the functionality of this block in IL/ST, please use the function block SET\_BITX (See *SET\_BITX: Set expanded bit*, p. 59).

## Representation

---

### Symbol

Block representation:



### Parameter description

Description of the block parameters:

Parameter	Data type	Meaning
IN	BOOL	Input data
NO	UINT	Bit number to be written.
RES	WORD	Output

---



---

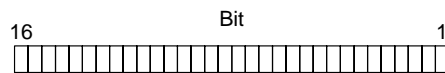
## Detailed description

---

**Function  
description**

The SET\_BIT function block sets the bit of the "RES" output selected with "NO" to the value of "IN".

The "NO" parameter provides the bit number in the output data.



SET\_BIT: Set bit

---

---

## SET\_BITX: Set expanded bit

13

---

### Overview

#### Introduction

This chapter describes the SET\_BITX block.

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	60
Representation	60
Detailed description	61


---

## Brief description

---

### Function description

This Function block sets the bit of the "RES" output register selected with "NO" to the value of "IN".

	<b>WARNING</b>
	<p><b>Dangerous process conditions</b></p> <p>This function block overwrites the values in status RAM WITHOUT regard to possible forced values in the reference data editor. This can produce dangerous process conditions.</p> <p><b>Failure to observe this precaution can result in severe injury or equipment damage.</b></p>

The parameters EN and ENO can additionally be projected.

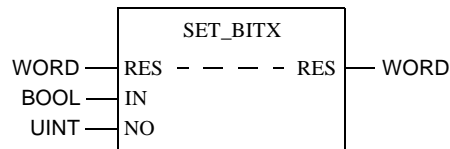
**Note:** This function block (in contrast to the SET\_BIT function block) can be used in programming languages ST and IL.

## Representation

---

### Symbol

Block representation:



### Parameter description

Description of the block parameters:

Parameter	Data type	Meaning
RES	WORD	Input for IN_OUT variables
IN	BOOL	Input data
NO	UINT	Number of Bit to be written.
RES	WORD	Output for IN_OUT variables

---

## Detailed description

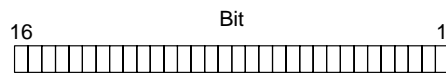
---

**Function description**

The SET\_BITX function block sets the bit of the "RES" output selected with "NO" to the value of "IN".

Since the input RS and the output RES concern an IN\_OUT variable, the same variable should be connected to the two parameters.

The "NO" parameter provides the bit number in the output data.



SET\_BITX: Set expanded Bit

---

---

## SRCH\_\*\*\*: Search

14

---

### Overview

#### At a glance

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

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	64
Representation	65
Detailed description	66

## Brief description

---

### Function description

This function block searches a source table for an entered bit pattern. In each cycle it searches the table for the next element, checks whether or not it matches the defined pattern and shows the result in a Boolean output. Then it increments the index in the source table for the next cycle.  
As additional parameters, EN and ENO can be configured.

**Note:** Since this function block has an output of data type ANY but has no input of this data type the ANY output must be assigned with the => operator. An instance of this function block may be called one time only. Calling the same function block instance more than once is not possible.

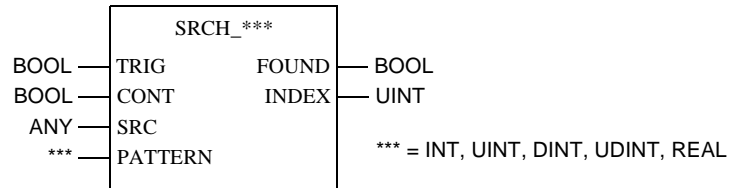
---



## Representation

### Symbol

Representation of the function block:



### Description of parameters

Description of block parameters:

Parameters	Data type	Meaning
TRIG	BOOL	TRIG detects the rising edge and starts the search.
CONT	BOOL	CONT defines whether to continue the search or to resume it at the beginning of the table after a pattern has been found.
SRC	ANY should be Array of INT, UINT, DINT, UDINT or REAL e.g. ARRAY[0..X] OF INT	Source table
PATTERN	INT, UINT, DINT, UINT, REAL	PATTERN is the bit pattern for the search.
FOUND	BOOL	1 = pattern found
INDEX	UINT	Once the pattern is found, the INDEX parameter shows where in the table the pattern has been found.

## Detailed description

---

### Function description

The SRCH function block searches a source table for an entered bit pattern. In each TRIG-enabled cycle it searches the table for the next element, checks whether or not it matches the defined pattern and shows the result in a Boolean output. Then it increments the index in the source table for the next cycle.

SRC (source) is type ANY which implicates a pre-defined length. This field (array) is interpreted as an ARRAY with the same data type (INT, UINT, DINT, UDINT, REAL) as the selected function block (refer to \_\*\*\*), this is independent of the actually selected data type for this pin (it could be a structure of different types).

TRIG detects the rising edge and starts the search for one cycle. The search will stop after this cycle until the next rising edge is detected at TRIG.

PATTERN is the bit pattern for the search. PATTERN has the same data type (INT, UINT, DINT, UDINT, REAL ) as the selected function block (refer to \_\*\*\*).

The CONT parameter defines whether to continue the search or to resume it at the beginning of the table after a pattern has been found.

Once the pattern is found, FOUND is set to "1" and the INDEX parameter shows where in the table the pattern has been found.

---

---

## T2T: Table to table

15

---

### Overview

#### At a glance

This chapter describes the T2T block

#### What's in this chapter?

This chapter contains the following topics:

Topic	Page
Brief description	68
Representation	69
Detailed description	70

---

## Brief description

---

### Function description

This function block copies the value from the parameter SRC, which is interpreted as a table to the parameter DEST, which is also interpreted as a table. As additional parameters, EN and ENO can be configured.

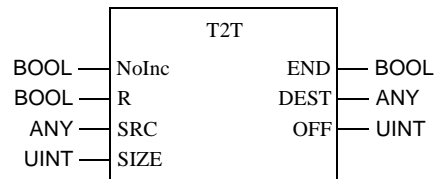
**Note:** Since this function block has an output of data type ANY bus has no input of this data type the ANY output must be assigned with the => operator. An instance of this function block may be called one time only. Calling the same function block instance more than once is not possible.

---

## Representation

### Symbol

Representation of the function block:



### Description of parameters

Description of block parameters:

Parameters	Data type	Meaning
NoInc	BOOL	1: freezes the pointer value
R	BOOL	1: resets the pointer value to zero
SRC	ANY should be Array ANY_ELEM e.g. .ARRAY[0..X] OF INT	Source data to be copied in the current cycle
SIZE	UINT	Size tells the function block how many words have to copied every cycle.
END	BOOL	1: Pointer value = table length (function block cannot increment any further)
DEST	ANY should be Array ANY_ELEM e.g. .ARRAY[0..X] OF INT	Destination register where source data will be copied in the cycle.
OFF	UINT	0: the parameter R is 1 before the copy is performed.

## Detailed description

---

### **Mode of functioning under Concept**

This function copies the value from the parameter SRC, which is interpreted as a table to the parameter DEST, which is also interpreted as a table.

The parameter OFF points into both tables. It is an index for the source array and the destination array where the source value should be copied from and to.

In each cycle the function copies the value of SRC[OFF] to DEST[OFF].

The offset will be incremented in each cycle by the number of 16bit words, which are copied, unless the parameter NoInc is 1. OFF is of the read/write type, equivalent to the VAR\_IN\_OUT IEC parameters

A parameter SIZE tells the EFB how many words have to be copied every cycle.

OFF is reset to 0 if the parameter R has the value 1 before copying.

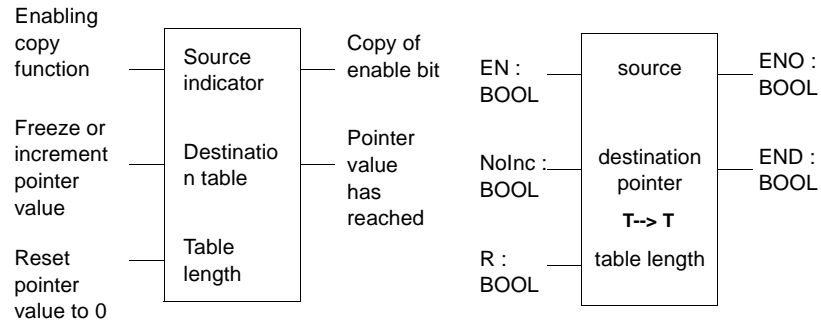
SRC and DEST are of type ANY which implies a pre-defined length. These variables will be interpreted as an ARRAY of bytes regardless of the type definition for these parameters (they may be a structure of different types).

OFF undergoes a bounds check each cycle. If OFF exceeds the length of one of the tables then END is set to 1 and OFF is not incremented. The function will not copy any more data until OFF returns within its bounds.

---

### Mode of functioning under Modsoft

This function copies a value from a source table to a destination table.



The table starts with the source register. The value in the destination pointer points to the offset in the table, where as many elements (words) as described in table length should be copied.

The value(s) are copied from the source register + offset + 1 to the destination register + offset + 1.

In each cycle the function copies as many 16-bit values from the source register to the destination register as defined in table length.

The offset is incremented in each cycle when the mid input is 0.

The offset is reset to 0 when the lower input is 1.

Entry, Input/Output	Meaning
Upper entry	Source is a reference into State-Ram: 0x, 1x, or in a contiguous 16-bit field 3x, 4x
Mid entry	Destination 'pointer' is reference into State-Ram: Start of 4x register + 1 + value in pointer
Lower entry	Length of table (1..255, 1..999, depending on processor)
Upper input, upper output	Enable function
Mid input	Incrementing offset for output table or not (0 or 1)
Lower input	Resetting offset for output table (if 1 then offset = 0)
Mid output	Offset reaching table size – no further copy operations

### Differences

Differences between Concept and Modsoft:

- The table is any pre-defined data structure. Modsoft functions cannot work with unlocated variables or links. Modsoft uses offsets in the state RAM. The table is part of the state RAM (not reserved, no checking for multiple use).
- Pins are named under Concept.
- EN/ENO are optional under Concept (Standard IEC1131-3).
- Various function names ('->' can not be used in a name under Concept).
- Various displays (see *Mode of functioning under Modsoft*, p. 71).

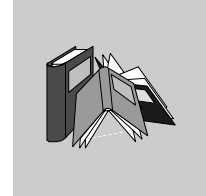
T2T: Table to Table

---



---

## 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>Base 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**

<b>Data transfer settings</b>	Settings, which determine how information from the programming device is transferred to the PLC.
<b>Data types</b>	<p>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".</p> <ul style="list-style-type: none"><li>• ANY_ELEM<ul style="list-style-type: none"><li>• ANY_NUM<ul style="list-style-type: none"><li>• ANY_REAL (REAL)</li><li>• ANY_INT (DINT, INT, UDINT, UINT)</li></ul></li><li>• ANY_BIT (BOOL, BYTE, WORD)</li><li>• TIME</li></ul></li><li>• System data types (IEC extensions)</li><li>• Derived (from "ANY" data types)</li></ul>
<b>DCP I/O station</b>	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.
<b>DDE (Dynamic Data Exchange)</b>	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, ò) 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 ( _ ) 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, discretes, 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>	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.  Example t#14MS, T#14.7S, time#18M, TIME#19.9H, t#20.4D, T#25H15M, time#5D14H12M18S3.5MS
<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 $2^{\text{exp}(32)}-1$ .
<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 $(2^{\text{exp}16})-1$ .
<b>Unlocated variable</b>	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.  Signals requiring no peripheral access, e.g. intermediate results, system tags etc, should primarily be declared as unlocated variables.

---

**V**

<b>Variables</b>	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.
<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

---



### B

Block Move, 37

### C

Concept EFBs and parameters, 16

Concept EFBs and the ANY data type, 17

### D

Data event logging for PCMCIA Read/write support, 21

DLOG, 21

### F

FIFO, 27

First-In/First-Out stack register, 27

Function

    Parameterization, 9

Function block

    Parameterization, 9

### G

GET\_3X, 31

GET\_4X, 33

GET\_BIT, 35

### I

IEC\_BMDI, 37

Implementation aspects, 17

### L

Last-In/First-Out stack register, 45

LIB984

    DLOG, 21

    FIFO, 27

    GET\_3X, 31

    GET\_4X, 33

    GET\_BIT, 35

    IEC\_BMDI, 37

    LIFO, 45

    PUT\_4X, 49

    R2T\_\*\*\*, 51

    SET\_BIT, 55

    SET\_BITX, 59

    SRCH\_\*\*\*, 63

    T2T, 67

Lib984, 13

    At a glance, 13

    DLOG, 21

    FIFO, 27

    GET\_3X, 31

    GET\_4X, 33

    GET\_BIT, 35

    IEC\_BMDI, 37

    LIFO, 45

    PUT\_4X, 49

    R2T\_\*\*\*, 51

SET\_BIT, 55  
SET\_BITX, 59  
SRCH\_\*\*\*, 63  
T2T, 67  
LIFO, 45

## M

### Modsoft Functions

Concept EFBs and Parameters, 16  
Concept EFBs and the ANY data type, 17  
Conversion in Concept, 15  
Differences in Concept, 15  
Implementation aspects under Concept,  
17  
Tables under Concept, 16  
Using the state RAM, 15

## P

Parameterization, 9  
PUT\_4X, 49

## R

R2T\_\*\*\*, 51  
Read bit, 35  
Reading 3x register, 31  
Reading 4x register, 33  
Register to table, 51

## S

Search, 63  
Set bit, 55  
Set expanded Bit, 59  
SET\_BIT, 55  
SET\_BITX, 59  
SRCH\_\*\*\*, 63

## T

T2T, 67  
Table to table, 67  
Tables under Concept, 16

## W

Write 4x register, 49