Programming Manual

BPRO3

Version 3.2

Doc. no. 212.952/DGB 04.95

Ident. no.: 00441108330

Edition: d131 April 95

SIG Positec BERGERLAHR GmbH&Co.KG

Breslauer Str. 7
Postfach 1180

D-7630 Lahr

Proposals
Improvements

BPRO3
Programming Manual

Edition: d131 April 95
Doc. no. 212.952/DGB 04.95

Sender:

Name:

Company/department:

Address:

Telephone no.:

Please inform us, using this form, if you have discovered any errors when reading this document.

We should also appreciate any new ideas and proposals.

Proposal and/or improvements:

Table of contents

Table of contents

Table of contents

1 General description

The BPRO3 programming system has been designed for creating, testing and documenting control programs for BERGER LAHR Series 300 controllers (e.g. WDP5-318). Programming of the controller is governed by the guidelines of the IEC 1131-3 standard.

IEC 1131-3 standard

Part 3 of the IEC 1131 standard (IEC 1131-3) covers:

- Program organization (blocks)
- Control of program execution (tasks)
- Programming languages, or means of program representation (e.g. IL)
- Program data (variable declaration, data types, data access)

A special feature of BERGER LAHR Series 300 controllers is the integration of PLC and movement functions within one control program.

1.1 Reference documentation

BPRO3 Programming Manual

The BPRO3 Programming Manual contains all information required for developing a control program.

BPRO3 Operating Manual

The BPRO3 Operating Manual contains information on installation and operation of the BPRO3 programming system and the various editors.

BPRO3 Library

The BPRO3 Library documentation describes the user library. The user library contains blocks which facilitate programming.

Controller manual

The controller manual contains controller-specific information.

1.2 Basic programming information

1.2.1 System environment

The programming device used is an IBM PC/AT (or compatible) computer with the BERGER LAHR BPRO3 programming system installed (for more information on program installation, see BPRO3 Operating Manual). The BNET network (a special BERGER LAHR network type) can be used for programming and operating up to 124 BERGER LAHR Series 300 controllers from the PC. The following are some typical characteristics of BERGER LAHR Series 300 controllers:

BNET network

- Programming according to IEC 1131-3
- Parallel PLC and movement function processing
- Individual movement programming due to various axis operating modes (point-to-point mode, speed mode, position following mode)
- Numerous predefined functions for movement programming

NOTE
Support for standardized serial interfaces (e.g. Profibus-DP) is in preparation.

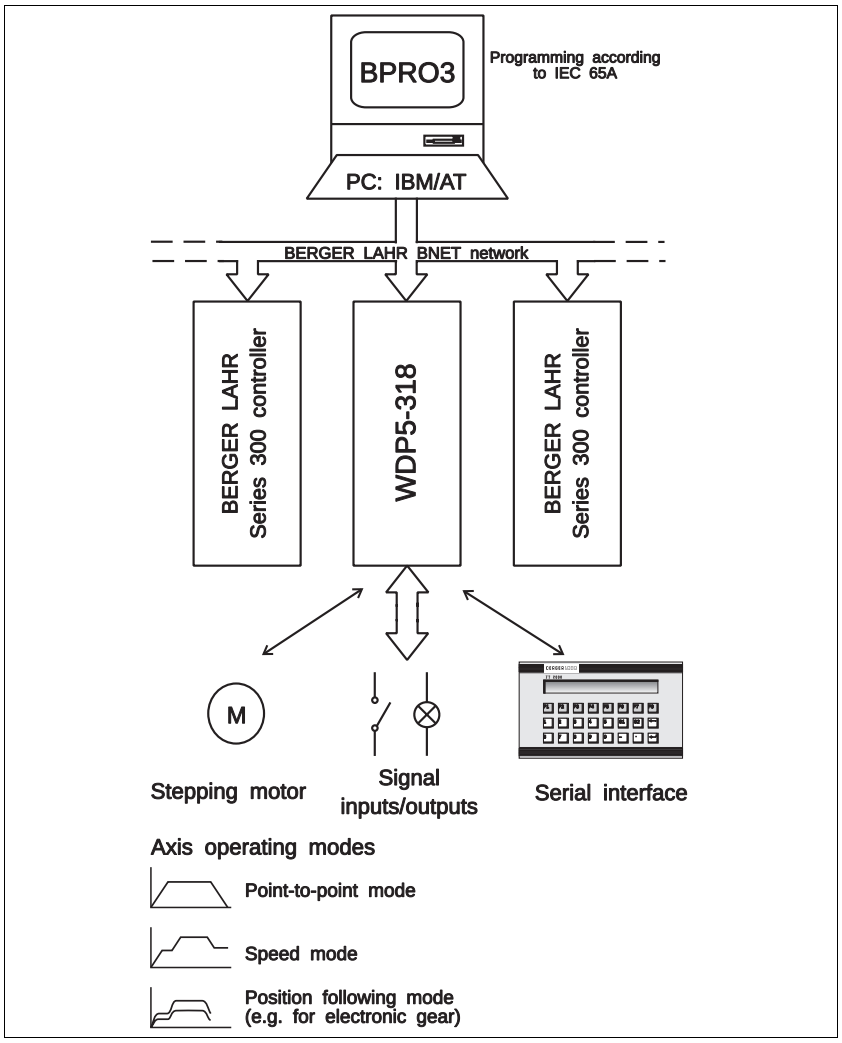


Fig. 1-1 System environment

1.2.2 Programming procedure

The aggregate of all information pertaining to a control program is called a project. A project requires the following information:

– **Assignment list**
A list of symbolic names for inputs, outputs and flags.

– **Hardware configuration (controller configuration)**
Information on the hardware and the type of controller which is designed to execute the program.

– **Software configuration (task configuration)**
Information on the point of time and the way program blocks are processed.

– **Program blocks**
Program components used for organizing a control program in a modular structure.

Figure 1-3 illustrates the programming procedure for creating a control program with the BPRO3 programming system.

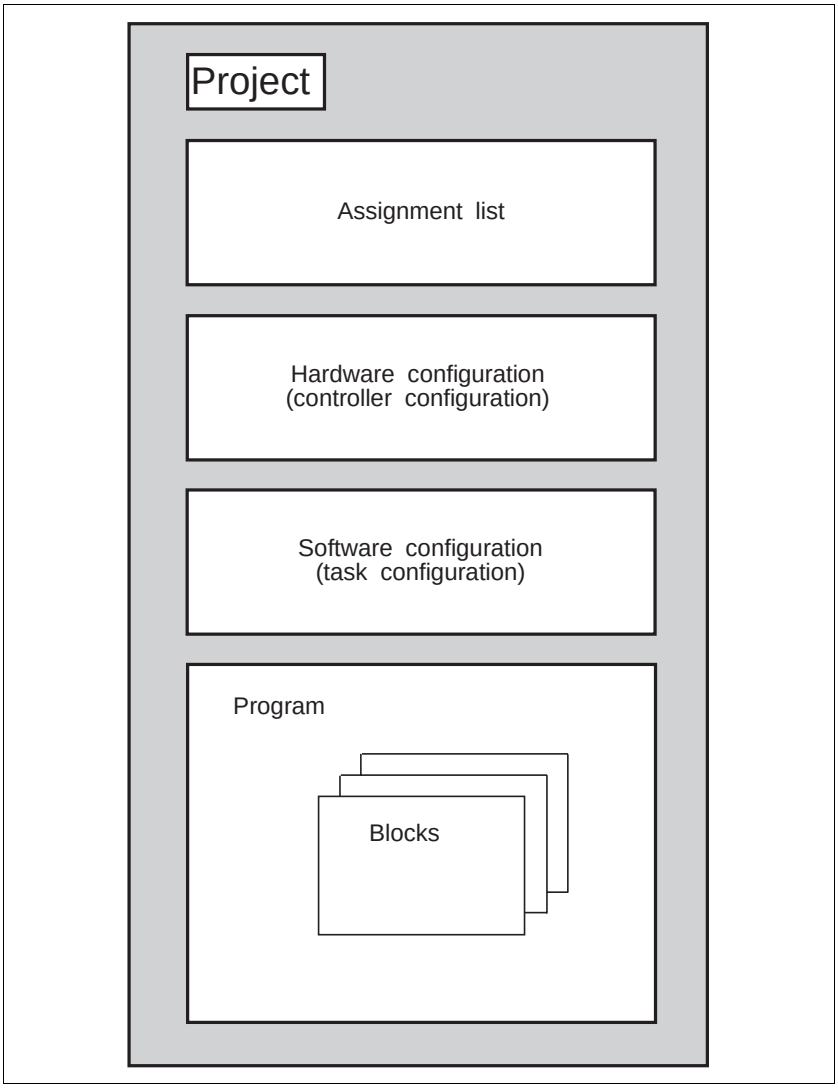


Fig. 1-2 Project components

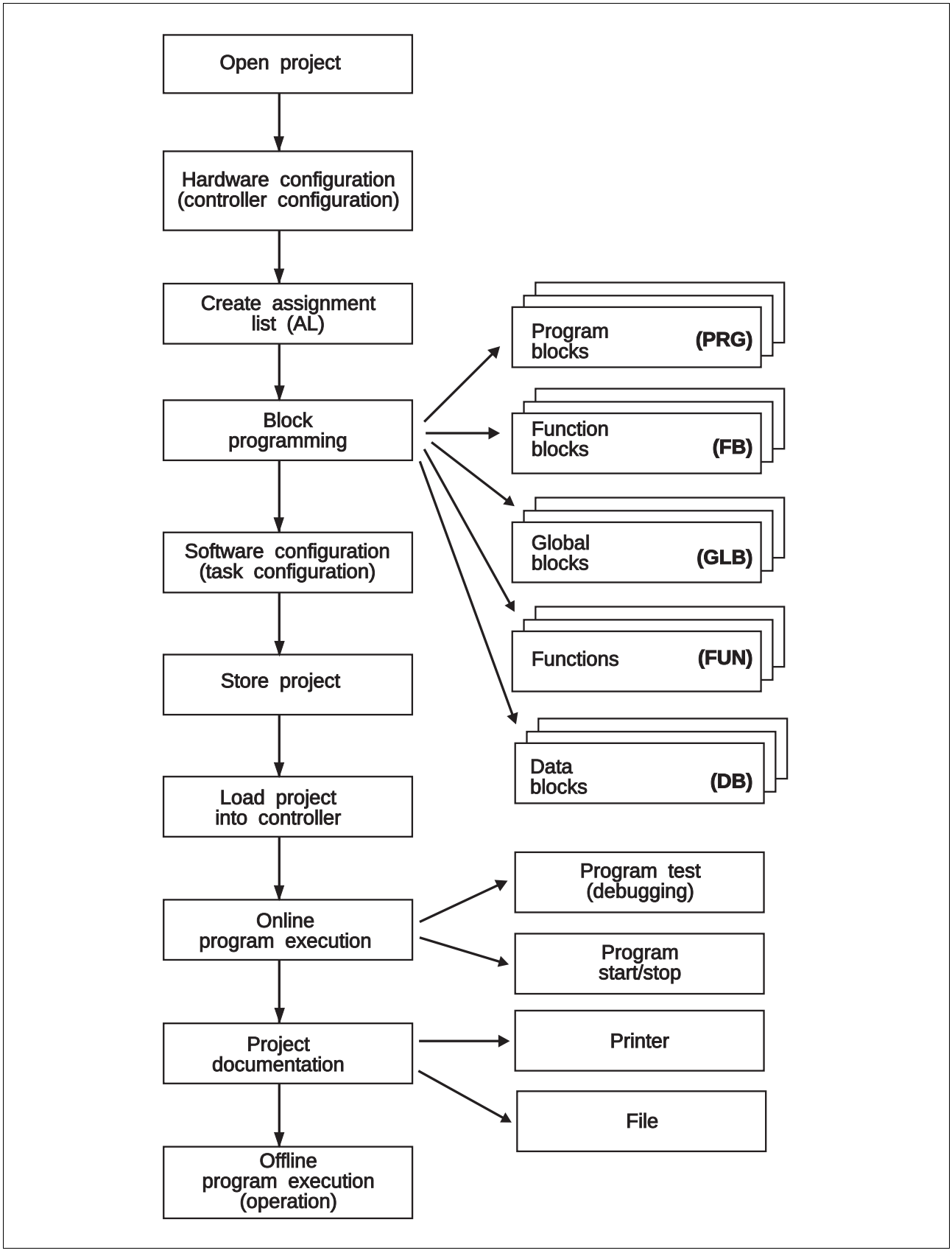


Fig. 1-3 Programming procedure

1.2.2.1 Open/store project

Working files

In order to create a project you first have to select “Open project”, i.e. select a project directory where the project-related files will be stored. The programming system creates working files in this directory which are modified during programming. At the end of a programming session the project must be stored, i.e. the working files are saved. You can revise the saved files later by selecting “Open project” to edit them and by selecting “Store project” to overwrite them.

1.2.2.2 Specify hardware configuration

Controller configuration editor

When you develop a control program you first have to inform the programming system about the hardware configuration (controller configuration) . For this purpose, the programming system includes a controller configuration editor. The controller configuration describes the type and the hardware components of the controller which is to execute the program. This enables the programming system to check the compatibility of program and controller.

1.2.2.3 Create assignment list

Standard symbols

Each project comprises an assignment list which is created using the assignment list editor (“AL editor”). In the assignment list, the inputs, outputs and flags can be assigned arbitrary names (e.g. inp1=%IX0.1). The data can be accessed in the program using either these names or the standard symbols (input = “I”, output = “Q”, flag = “M”). Inputs, outputs and flags do not necessarily have to be assigned names, however, this offers the following advantages:

- Improved readability of the program
- When modifying the input/output wiring you only have to modify the assignment list, not the program itself.

1.2.2.4 Create blocks

A control program consists of several program components (blocks) which are used for organizing the program in a modular structure. The following block types are available:

- Program blocks (PRG)
- Function blocks (FB)
- Global blocks (GLB)
- Functions (FUN)
- Data blocks (DB)

NOTE
The individual blocks of a program can also be used in other programs. Furthermore, predefined blocks are available in the “library” of the BPRO3 programming system. For more information on blocks, see chapter 1.2.3.1.

Block header and block body

Instructions

Blocks consist of a block header and a block body. The block header includes information on the block itself (such as name, type, author, etc.) as well as declarations of variables and possibly function blocks. The block body contains instructions for the control program. The block header is created using the “block header editor”, the block body with the “IL editor” (in a future version also using the “FB editor” or the “SFC editor”).

1.2.2.5 Specify software configuration

Task

A control program is composed of various program components (program blocks) which must be assigned to a task. A task determines the way program components (program blocks) are processed in the controller. At present, three task types are implemented:

- INIT task to initialize the controller
- PLC task to execute program blocks in cycles
- SEQUENCE task to execute program blocks in steps.

Task configuration editor

Task configuration list

The task configuration editor can be used for assigning one or more program blocks to a task. If several program blocks are assigned to a task, they are processed in the order of the entries in the task configuration list (from top to bottom).

1.2.2.6 Load project into controller

Within the context of the BPRO3 programming system, loading a project into the controller is called a “project download” .

Download project

When “downloading” a project or individual blocks, the blocks are automatically compiled and linked (if not already done before downloading).

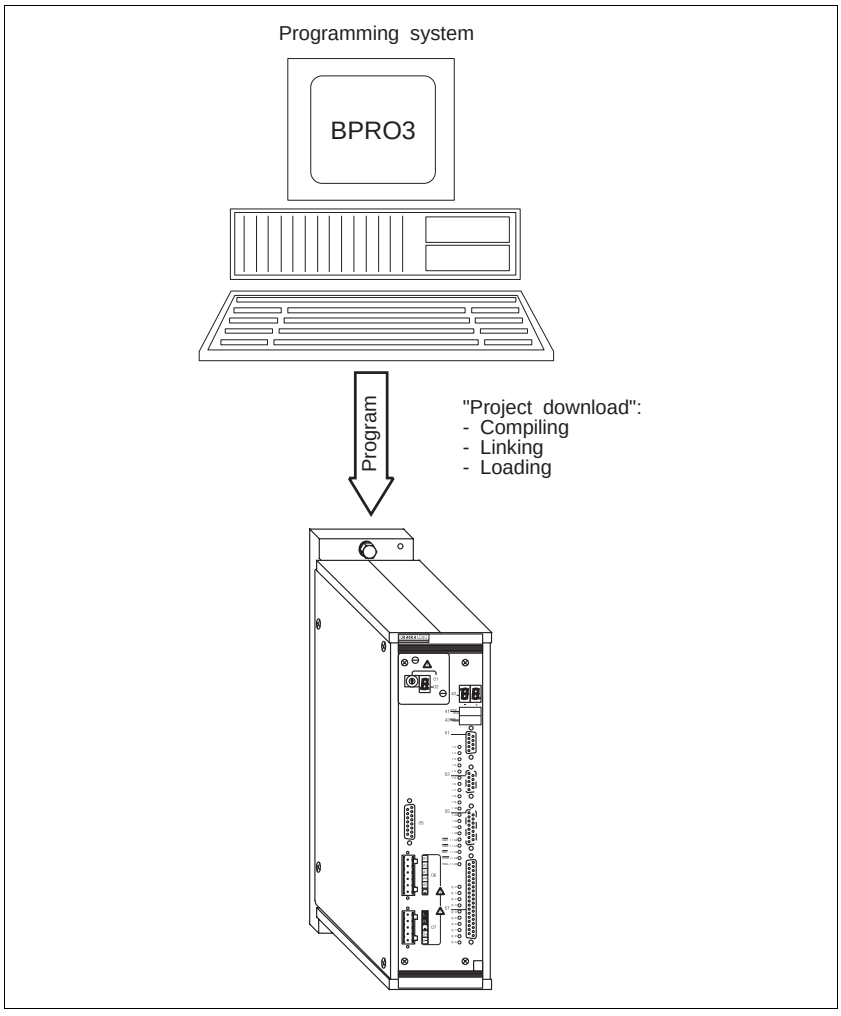


Fig. 1-4 Loading project into controller

Upload project

Loading a project from the controller into the programming system is called a “project upload” .

NOTE
Individual blocks or tasks can also be downloaded or uploaded; see BPRO3 Operating Manual.

Compiling

Storage format (source code)

The blocks are stored for editing in the programming system using a defined format (source code). When downloading a project or individual blocks, the blocks are translated (compiled) from source code into an executable code: “object code” or “pseudo-code”.

Object code

– “Object code”
Instructions compiled into object code can be directly executed on the controller. Object code instructions are faster in execution than pseudo-code instructions.

Pseudo-code

– “Pseudo-code” or “intermediate code”
Instructions loaded as pseudo-code into the controller can be decompiled into source code (not yet implemented), which means they can be re-edited (this is not possible with plain object code). Pseudo-code instructions must be interpreted before being executed on the controller. To test (“debug”) a program or block, the instructions must be loaded in pseudo-code.

Testing (debugging)

NOTE
It is also possible to load blocks in object code and pseudo-code. In this case, the instructions are executed in object code and can still be decompiled into source code.
In addition, pseudo-code can be used to store symbols (block names, variable names, etc.) and/or symbols and comments. The actual pseudo-code does not include any symbols or comments. Instead of using symbols, the programming system assigns numbers during an upload.

Linking

Linking

In order for a program to run on the controller, links must be created between the individual program components. This process is called “linking” and is carried out automatically during a download.

1.2.2.7 Program execution

When you have loaded the project into the controller you can start or stop the program:

- from the programming system (on-line) or
- from the controller front panel (off-line).

Debugging

You can also test (debug) the program via the programming system. For more information, see BPRO3 Operating Manual.

NOTE
To test the program, the program must be loaded into the controller in pseudo-code. You should always store a tested and debugged program in object code since program execution is considerably faster in object code.

1.2.2.8 Program documentation

Documentation file (.DOC)

On completion of the project the project files can be output (in the screen display formats of the various editors) to the printer or to a documentation file (.DOC). You can specify previously which information is to be output.

1.2.3 Control program structure

1.2.3.1 Block types

Organizing
Modular structure

A control program consists of several program components (blocks) which are used for organizing the program in a modular structure. The individual blocks of a program can also be used in other programs. In addition, predefined blocks are available in the library of the BPRO3 programming system; see chapter 7.

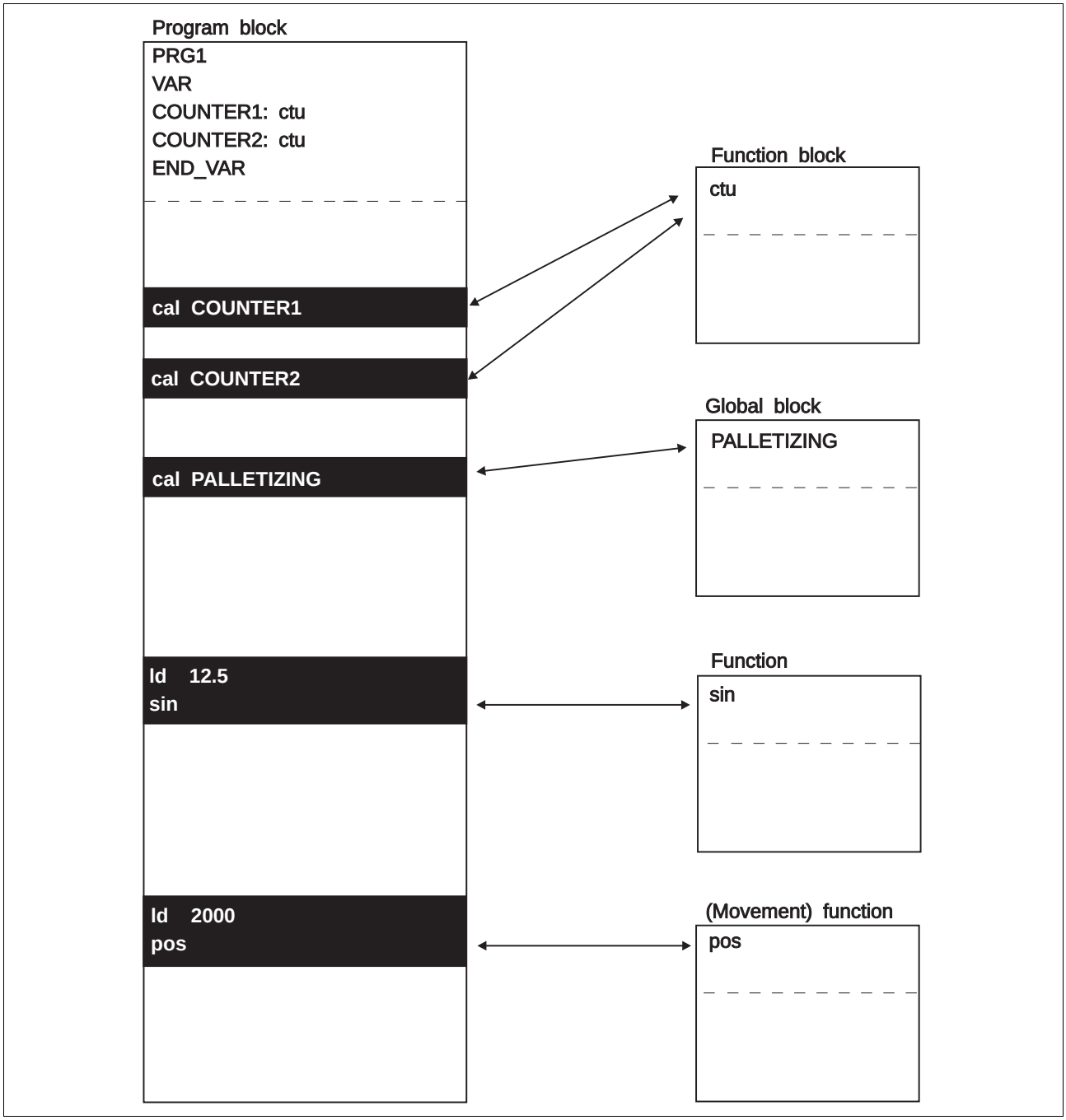


Fig. 1-5 Block types

The following block types are available:

Program blocks (PRG)

Program components

Program blocks are program components which can be assigned to a task. Data exchange between the program blocks of a task is effected through global variables (e.g. flags); see chapter 1.4.2.2. Within a program block, function blocks, global blocks and functions can be called.

Data exchange

Global blocks (GLB)

Global blocks can be called from program blocks, function blocks and other global blocks. They contain input/output variables for passing data to the calling block. Global blocks can be used for storing status conditions, i.e. the variables retain their values after the block has been processed. This means that not all input parameters must be specified when calling a global block. Within a global block, other global blocks, function blocks and functions can be called.

Input/output variable

Function blocks (FB)

Function blocks have the same characteristics as global blocks, however, they have to be declared in the block headers of the calling blocks, i.e. a function block type is assigned a name and allocated memory space for variables. This allows you to use several function blocks of the same type but with different names and allocated memory within a program or block; see figure 1-6.

Function block type

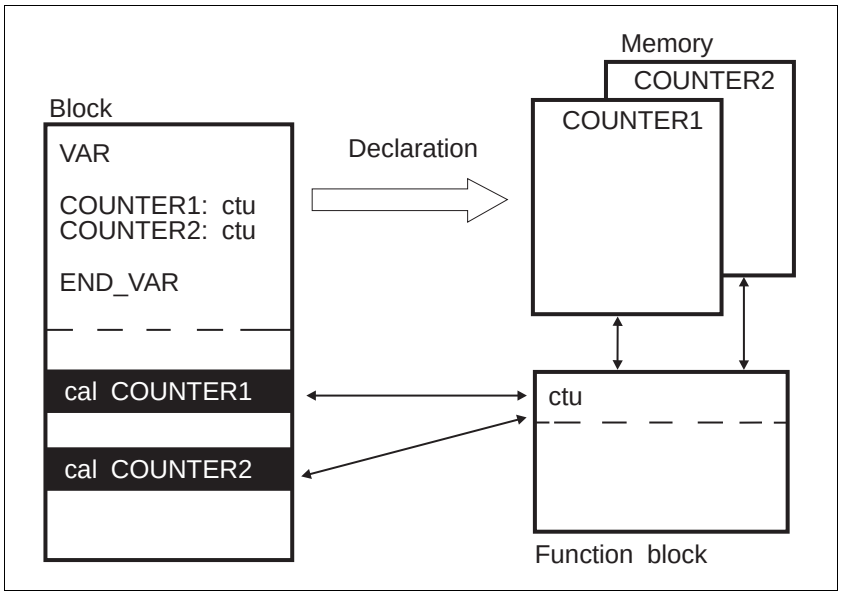


Fig. 1-6 Function block declaration

Functions (FUN)

Current result

Function call

Functions (such as “sin”) can be used by any block. A function can have several input variables (in a function call, the first input variable is always the CR) but only one output variable. The result is stored in the output variable of the function and passed to the calling block in the current result CR (see chapter 1.3.1.2). The local variables of functions are reinitialized after each function call. The same input therefore always returns the same result. A function does not need to be declared and is always called with the same name.

NOTE
The library of the BPRO3 programming system includes predefined functions. Movement functions are a special function type within the library. Movement functions are used for controlling and operating the drive.

Data blocks (DB)

Data block type editor

Data structure

A data block is a special type of block. It does not contain any instructions but an arbitrary number of variables which can be accessed from each block (global variables). In order to create a data block, you first have to use the data block type editor to create a data block type. A data block type describes a data structure which contains several variable types. Several data blocks can be created from one data block type with the data block editor.

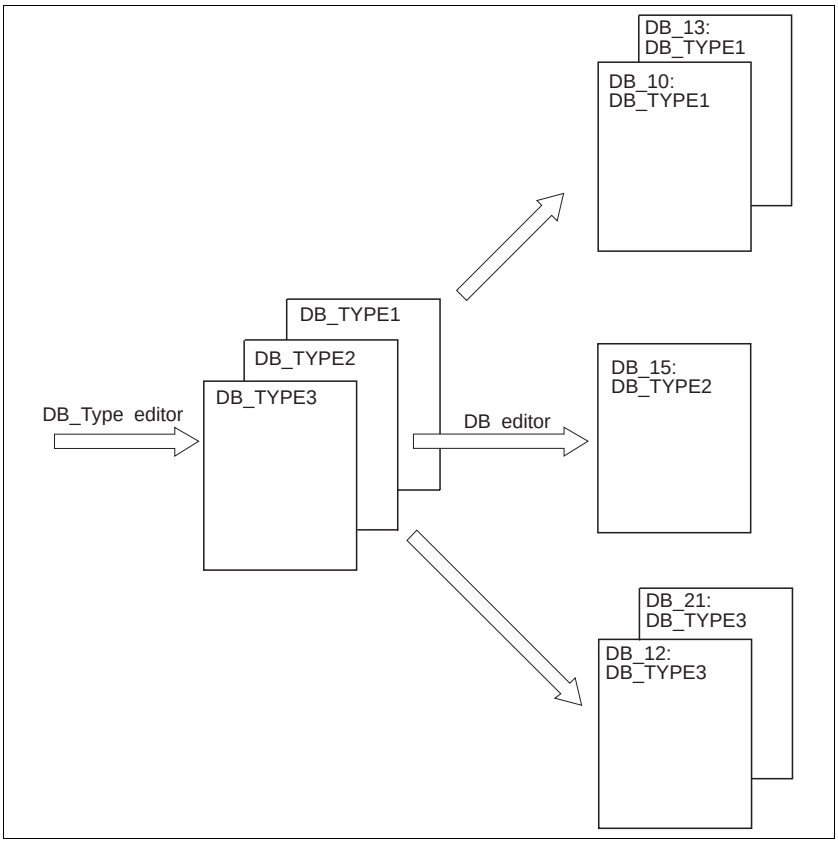


Fig. 1-7 Data blocks and data block types

1.2.3.2 Block structure

Each block, except data blocks, comprises a block header and a block body.

Block header

Block header editor

The block header contains information on the block itself (e.g. name, type, etc.) and variable declarations. Block headers are created using the block header editor. The comment editor can be used to insert comments into the block header.

Block body

IL editor

The block body contains program instructions written in the IL (instruction list) programming language. They are entered using the “IL editor” .

NOTE
In a future release of the programming system it will also be possible to use the programming languages “FBD” (function block diagram), “SFC” (sequential function chart) and “C” to create a program.

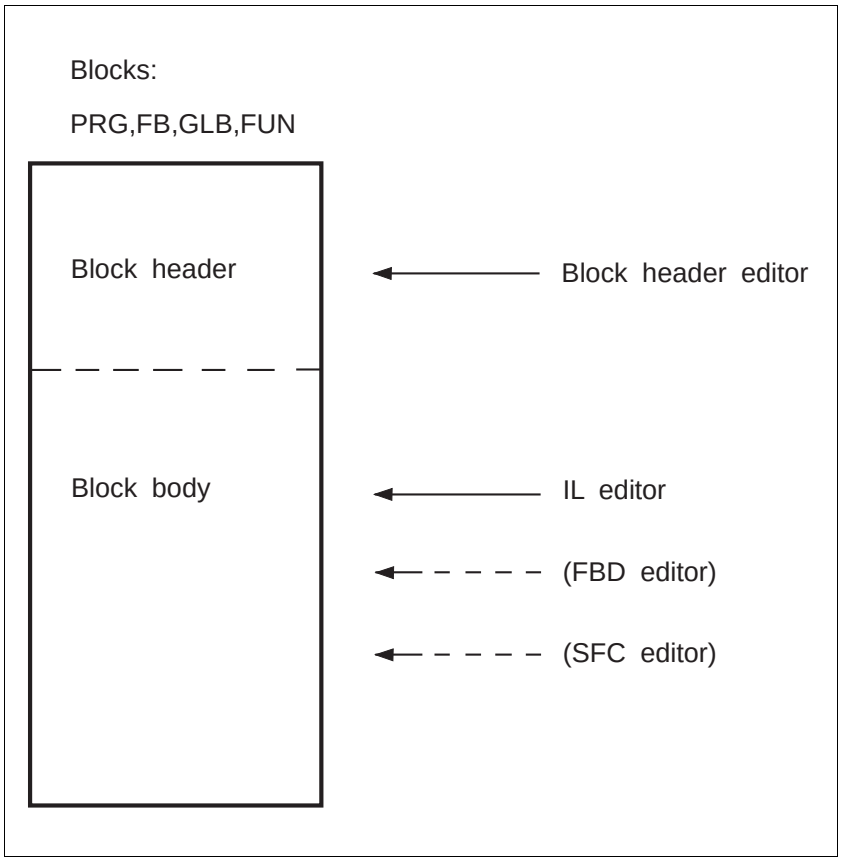


Fig. 1-8 Block structure

1.2.3.3 Block calling

Blocks are called in different ways, depending on the block type.

Block type	Call	Examples
Program block	According to task assignment (no explicit call is possible)	See task editor in BPRO3 Operating Manual
Global block	With the “cal” instruction and input parameters.  *NOTE* *When calling a global block not all input parameters must be specified since a global block has storage capability.*	cal gripper1 (value1:=inp1, value2:=100)
Function block	With the “cal” instruction and input parameters.  *NOTE* *When calling a function block, not all input parameters must be specified since a function block has storage capability.*	As for global blocks, however, the function block must be declared in the header of the calling block.
Function	With the function name as the operator and input parameters as operands. The value for the first input parameter and the result of the function are always contained in the CR.  *NOTE* *When calling a function, all input parameters must be specified in the order of their entries in the block header of the function.*	ld 1000 setpos x,actual
Data block	Access to variables via the data block name. Any data block used must have been declared with the data block editor.	ld dbl1.var10 add 100 st dbl.var2

1.2.4 Control program execution

Task configuration list

To execute a control program, tasks are required. Tasks are program organization units which are similar to the organization blocks of conventional programmable logic controllers. They control the way and the time of execution of program blocks. One or more program blocks can be assigned to a task. The program blocks of a task are processed in the order of their entries in the task configuration list (from top to bottom). The following task types are defined.

1.2.4.1 INIT task

Output initialization
Drive presettings

The INIT task controls the customer-specific default settings of the controller (e.g. output initialization, drive presettings, etc.). Program blocks assigned to the INIT task are executed once after program start. Other tasks cannot be executed until the INIT task has been processed.

1.2.4.2 PLC task

Process image

The PLC task is called after execution of the INIT task and is executed in cycles. When the PLC task starts, the inputs are read into the process image PI (temporary storage area for inputs/outputs). After this step the program blocks of the PLC task are processed sequentially. Subsequently the changes in the output status conditions in the process image are output and the cycle restarts.

NOTE
Input/output signals can also be accessed/changed directly in the program without accessing the process image.

The time required for one cycle is called the cycle time. The cycle time depends on:

- the length of the program
- the type of instructions
- the download formats (object code or pseudo-code) of the individual blocks.

Cycle time monitoring

Cycle time

During program execution, cycle time monitoring is performed. The cycle time monitoring feature detects any timeout with respect to the predefined maximum cycle time (factory setting 2 s). The library function “cycletime” can be used to set the maximum cycle time or disable cycle time monitoring.

1.2.4.3 SEQUENCE task

Time-slice principle

The SEQUENCE task is executed simultaneously with the PLC task (according to the time-slice principle), however, SEQUENCE task program blocks are processed only once. The process image is updated by the PLC task.

Execution time

The SEQUENCE task is suitable for programming straightforward step sequences, e.g. a series of positioning operations. The execution time of a SEQUENCE task is not monitored. It is recommended to use SEQUENCE tasks primarily for movement programming.

NOTE
The PLC task and the SEQUENCE task are processed according to the time-slice principle: The PLC program and the SEQUENCE program share the processing time, switching every 2 ms from one program to the other.
When the last SEQUENCE task block has been processed, control passes completely to the PLC task.
In the SEQUENCE task the inputs/outputs should be accessed directly since the process image is updated by the PLC task (see chapter 1.4.2.5).

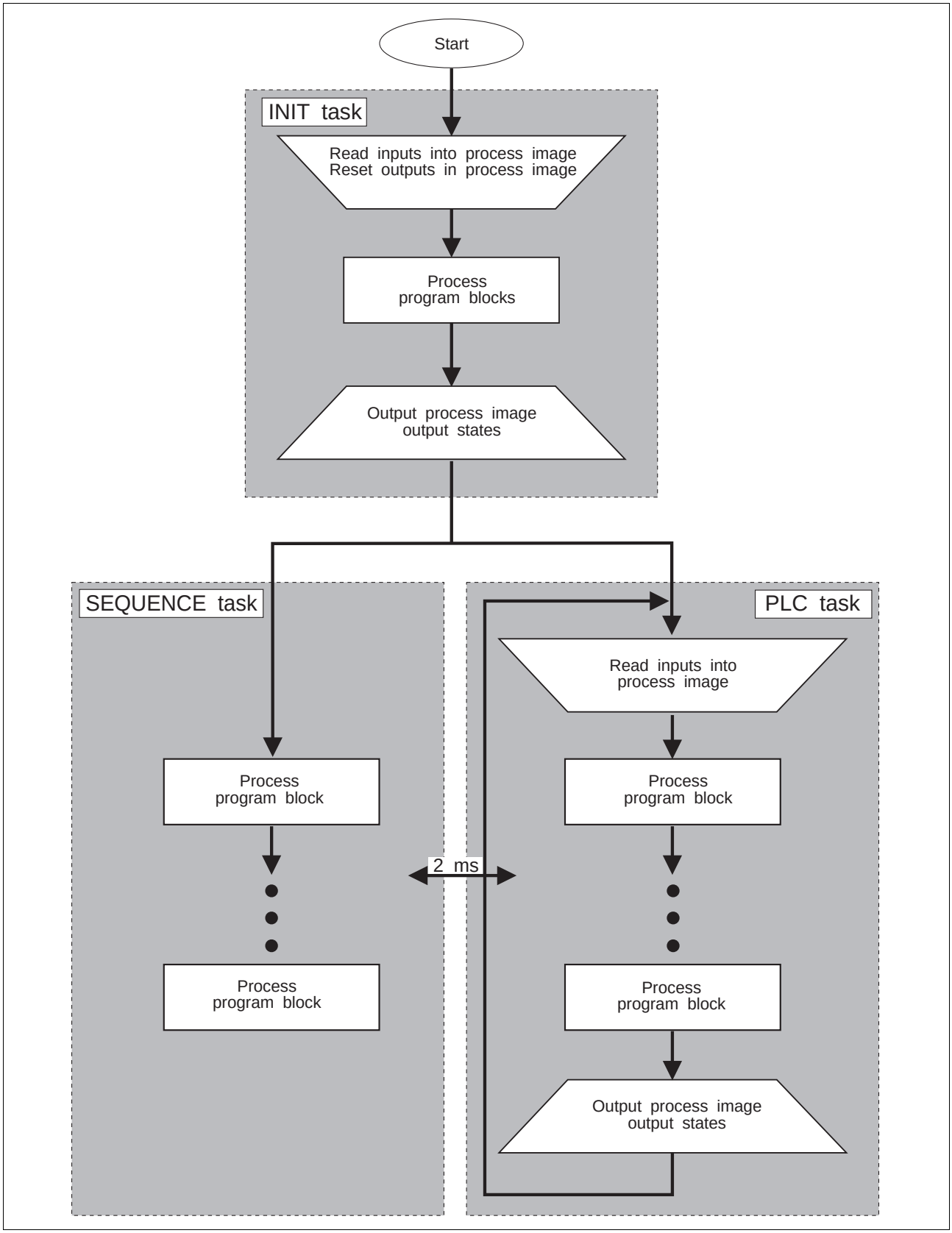


Fig. 1-9 Program execution

1.3 Programming languages

Program representation

The programming languages for programmable logic controllers differ essentially in their way of program representation (textual or graphic languages). The IEC 1131-3 standard specifies the following programming languages:

Textual representation

- Instruction List language (IL)
- Structured Text language (ST)

Graphic representation

- Function Block Diagram language (FBD)
- Sequential Function Chart language (SFC)
- Ladder Diagram language (LD)

At present, the IL programming language is implemented in the programming system. FBD and SFC will follow.

NOTE
Although the FBD programming language is not yet implemented in the programming system, it will be used in this documentation by way of simplified illustration of IL programming examples.

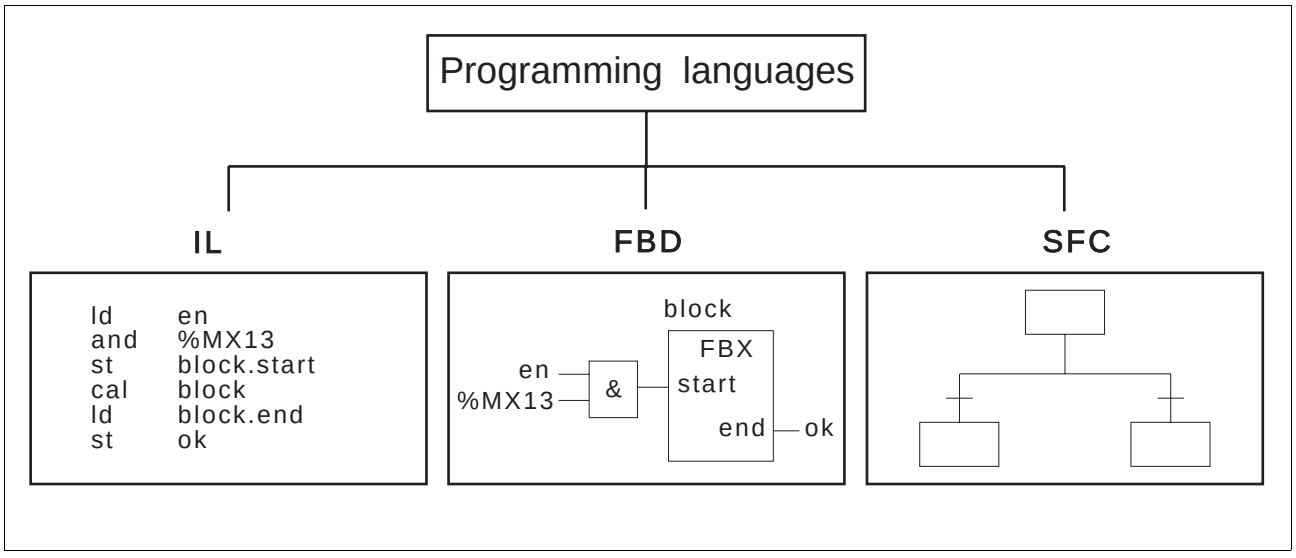


Fig. 1-10 Programming languages

1.3.1 Instruction List (IL)

The instructions for an IL program are entered and displayed as text.

1.3.1.1 Operator and operand

An instruction is made up of at least one operator and one or more operands.

Operator	Operand	
ld	15	(load value 15 into CR)

The operator describes the operation which is to be carried out with the operand and the current result (CR).

NOTE
In addition to the operator and the operand, an IL instruction may include a modifier; see the chapter on IL programming.

1.3.1.2 Current result (CR)

The current result (CR) is a temporary storage feature (accumulator) of the controller which is used for:

– Arithmetic operations
– Logical operations
– Passing data.

IL example		**Comment**
ld	15	Load value 15 into CR
add	10	Add 10 to CR
st	Sum	Store contents of CR in the Sum variable

NOTE
In contrast to conventional programmable logic controllers, BERGER LAHR Series 300 controllers only have one accumulator for the CR. The memory size of the accumulator is automatically adjusted to the data type of the operand.

1.3.1.3 IL networks

An IL program can consist of several “networks” . Such a “network” is a sequence of functionally related IL instructions. You can freely define how many instructions are combined to form a network. Each network can be complemented with a comment. Large comments can be entered by means of the comment editor.

Comment editor

At the beginning of an IL network, a label can be inserted which can be referenced in a “jmp” instruction.

NOTE
A network is represented in the FBD programming language as a sequence of interconnected graphic symbols; see figure 1-11. Programs created in IL (or FBD in the future) are also called network lists (NWL) in the programming system.

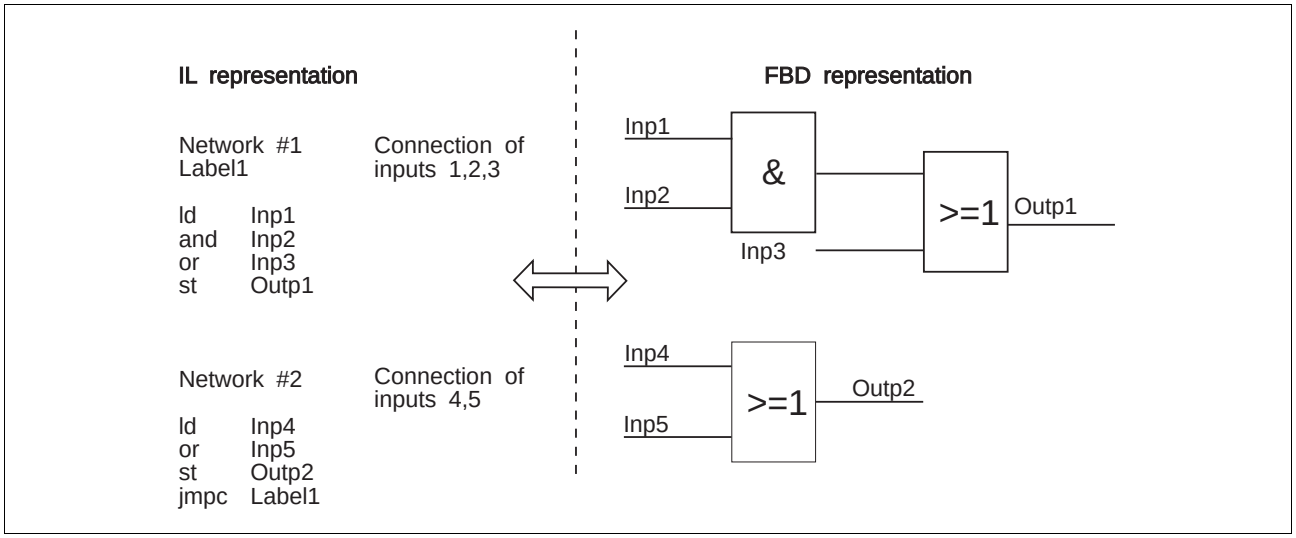


Fig. 1-11 Networks

1.4 Program data

1.4.1 Data types

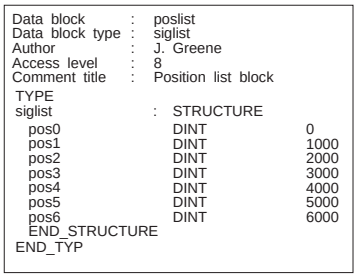
Memory space allocated

When declaring a variable, the data type of the variable must be specified. The data type determines the memory space allocated and the range of values of a variable. There are elementary and derived data types.

1.4.1.1 Elementary data types

Data type	Definition/memory requirement	Range of values	Input example
BOOL	Bit string of length 1/1 bit	0, 1	0 or FALSE 1 or TRUE
BYTE	Bit string of length 8/8 bits	0 to 255 (decimal)	93 2#01011101 16#5D
WORD	Bit string of length 16/16 bits	0 to 65535 (decimal)	16429 2#100101101011 16#5FA7
INT	Integer/16 bits	-32768 to +32767 (decimal)	-2456 2#10011001111 16#4C5B
DINT	Integer/32 bits	-2147483648 to +2147483647 (decimal)	2567890 2#101101101001110 16#B5A8
LREAL	Real number/64 bits	$\pm$ 9.46 x 10^{-308} to $\pm$ 1.79 x 10^{+308} (decimal)	12.345 4.67E-223 -1.267e48
TIME	Period/32 bits	Days (2 digits)d Hours (2 digits)h Minutes (2 digits)m Seconds (2 digits)s Milliseconds (3 digits)ms (internal time resolution = ms)	T#5d12h20m4s15ms T#12h_30m or T#12.5h

1.4.1.2 Derived data types

Data type	Meaning	Maximum size	Input examples
STRING	ASCII string	255 characters (bytes)	'This is the CR code: $0D'
ARRAY	Arrays of elementary data types	255 x 255 x 255 elements	1,10,5,7,58 for variable of type: ARRAY (0..4) OF INT ARRAY (0..4, 0..5) OF BOOL
Data block DB	Structure made up of elementary and derived data types	32 Kb	

```
Data block        :  poslist
Data block type   :  siglist
Author            :  J. Greene
Access level      :  8
Comment title     :  Position list block
 TYPE
 siglist                :  STRUCTURE
   pos0                    DINT          0
   pos1                    DINT          1000
   pos2                    DINT          2000
   pos3                    DINT          3000
   pos4                    DINT          4000
   pos5                    DINT          5000
   pos6                    DINT          6000
   END_STRUCTURE
 END_TYP
```

1.4.2 Variables

Variables are storage elements which are used in a program for data exchange and for data storage.

Declaration

Before you can use variables in a block you have to declare them. When declaring a variable, the access privilege is defined (see chapter 1.4.2.4). Depending on the access privilege of a block, the following variable types are defined:

- Local variables
- Global variables
- Input/output variables

1.4.2.1 Local variables

Local variables are storage elements which can be accessed only from within a block. Local variables are declared in the block header with VAR.

NOTE
A called block (e.g. function) cannot access the local variables of the calling block (e.g. program block).

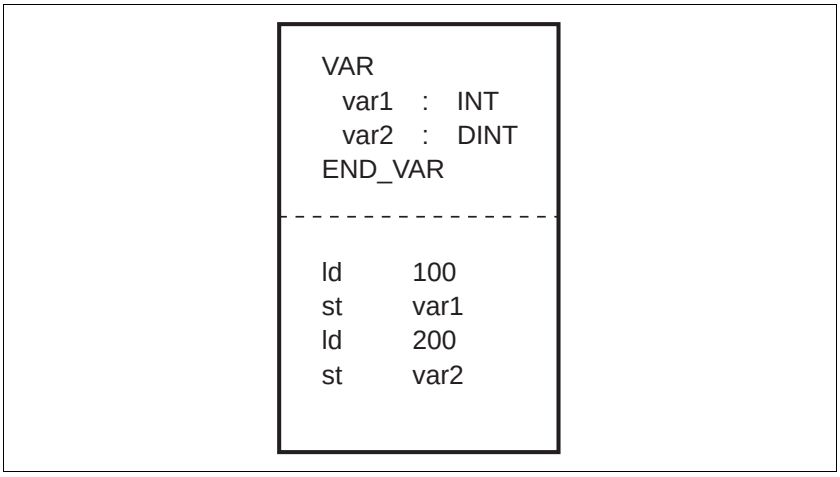


Fig. 1-12 Local variables

1.4.2.2 Global variables

Global variables are storage elements which can be accessed from any block.
The following global variables are defined:

- Signal inputs
- Signal outputs
- Flags
- Variables declared in data blocks

NOTE
The signal inputs/outputs and the flags can be assigned symbolic names in the assignment list (e.g. Inp1=%IX0.1). The variables of the assignment list must be declared as VAR_EXTERNAL before they can be used in the individual blocks.

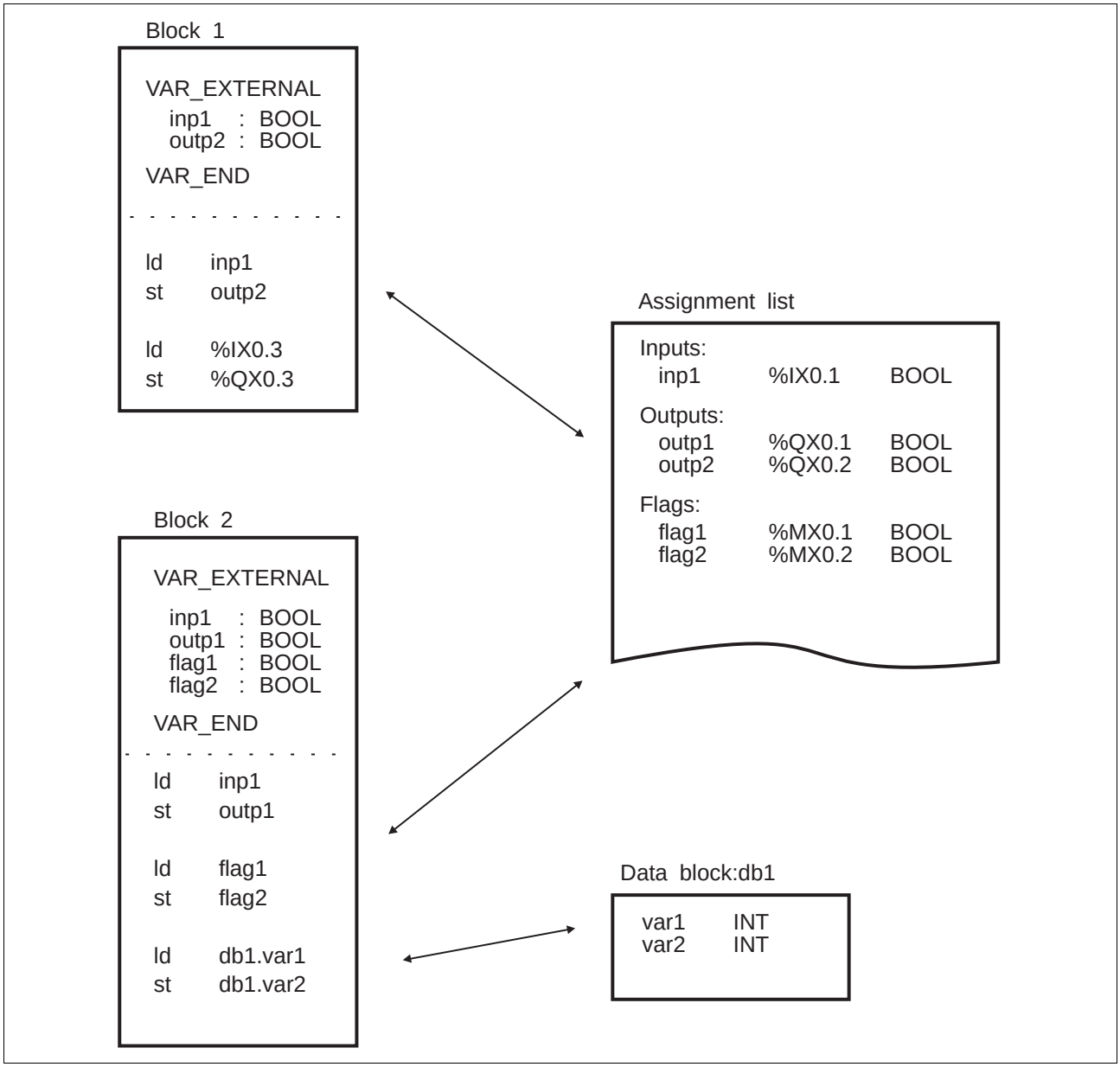


Fig. 1-13 Global variables

1.4.2.3 Input/output variables

Input/output variables are storage elements which are used for passing data from one block to another (e.g. between program blocks and function blocks). Input/output variables must be declared in the block header of the block to be called with VAR_INPUT, VAR_OUTPUT or VAR_IN_OUT.

NOTE
When a block is called, an input/output variable must be passed another variable.

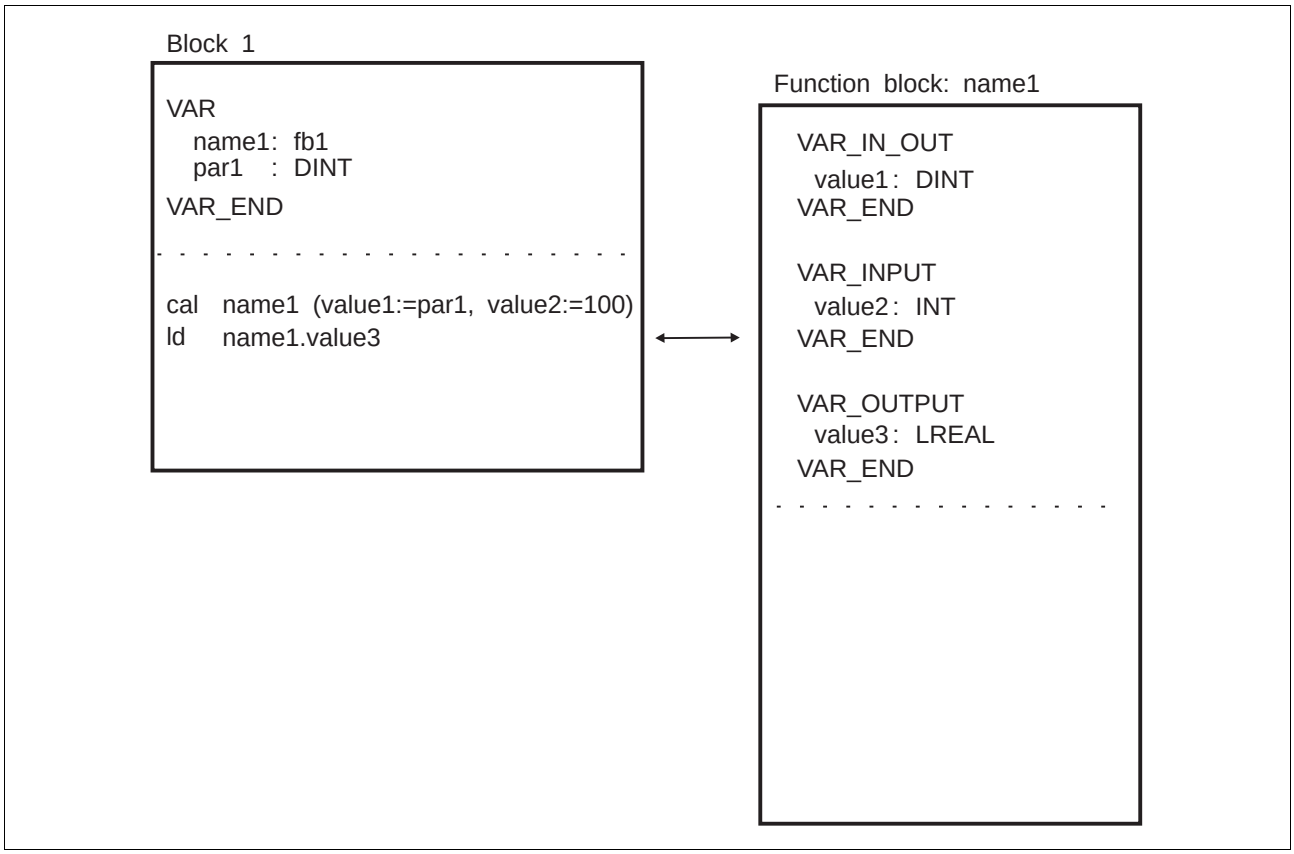


Fig. 1-14 Input/output variables

1.4.2.4 Flags

Flags are storage elements which can be accessed from any block. The controller has a dedicated memory area for flags; this area can be specified using the programming system (“controller configuration”).

Accessing flags

When accessing flags, a leading “%” character must be specified. A flag bit can be accessed with “X”, a flag byte with “B”, and a flag word with “W”. A flag bit is always of BOOLean data type, a flag byte of BYTE type, a flag word of WORD type.

Flag bit

Flag bits are accessed on a word basis, i.e. first you have to specify a word number and then a bit number. Flag words and flag bytes are only addressed with their word or byte number.

Addressing examples:

ld	%MX0.5	Load flag bit 5 of flag word 0 into CR
st	%MB2	Store contents of CR in flag byte 2
ld	%MW3	Load flag word 3 into the CR

ATTENTION
The high-order byte of a flag word always contains the low-order bits.

NOTE
Flags can also be referenced with symbolic names (see chapter 1.5) if these are defined in the assignment list and declared as VAR_EXTERNAL. A flag in the assignment list can be assigned any data type (e.g. DINT).

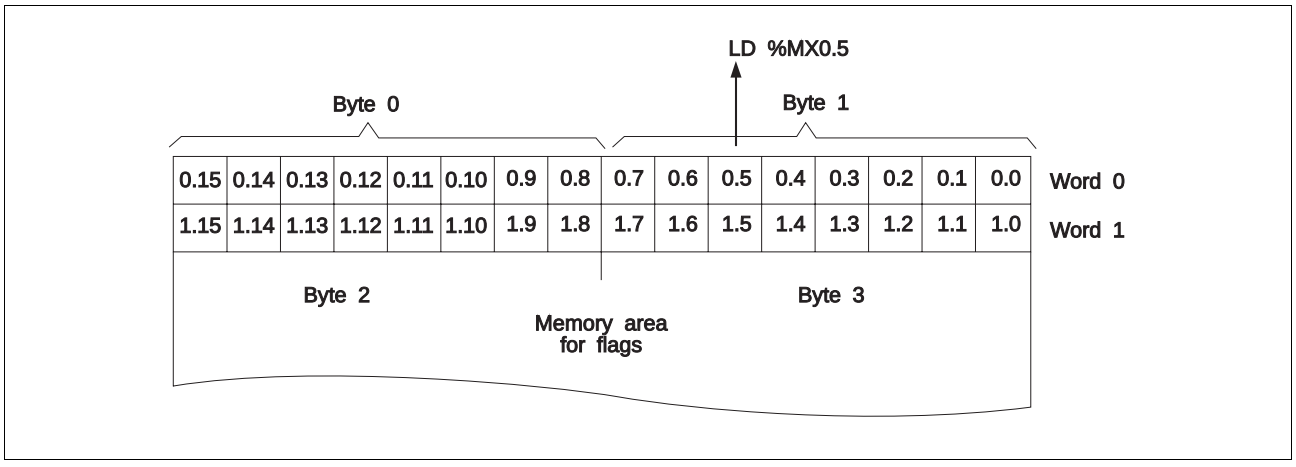


Fig. 1-15 Accessing flags

1.4.2.5 Inputs/outputs

Sequential operations

Process image PI

I/O module number

The controller has a certain number of inputs and outputs which are used to control sequential operations. Input/output processing can be effected simultaneously with the execution of movements.
The inputs/outputs can be accessed directly (with “@”) or indirectly via the process image PA (with “%”). You can access a single input/output bit (with “X”), an input/output byte (with “B”) or an input/output word (with “W”). When addressing inputs/outputs you have to specify the I/O module number, the word number and, if applicable, the bit number. (For information on the relationship between node/module numbers and the input/output modules of the controller, see BPRO3 Operating Manual, chapter 3.4.6, “Controller configuration”.)

Addressing examples:

ld	%IX0.10	Load input 10 of controller from process image into CR
ld	@IW0	Load input word 0 (inputs 0 to 15) directly into CR
st	%QX0.4	Write contents of CR to output bit 4 of process image

ATTENTION
The high-order byte of an input/output word always contains the low-order bits.

NOTE
Inputs/outputs can also be referenced with symbolic names (see chapter 1.5) if these are defined in the assignment list and declared as VAR_EXTERNAL.

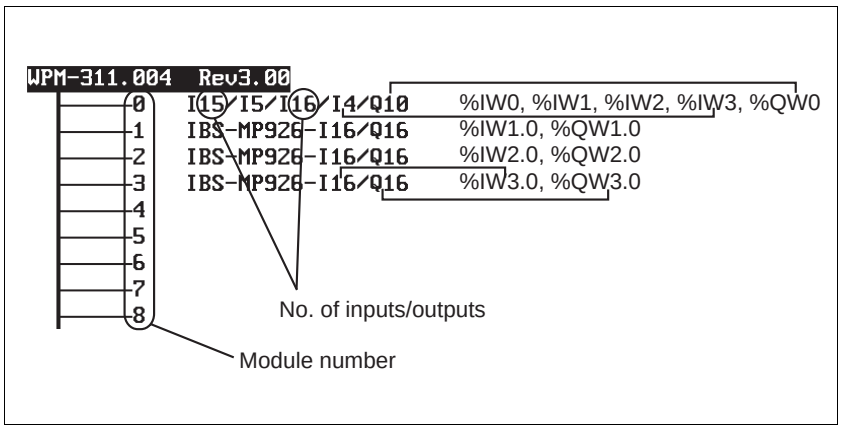


Fig. 1-16 Controller configuration in BPRO3

Accessing inputs/outputs using the WPM-311.004 unit as an example

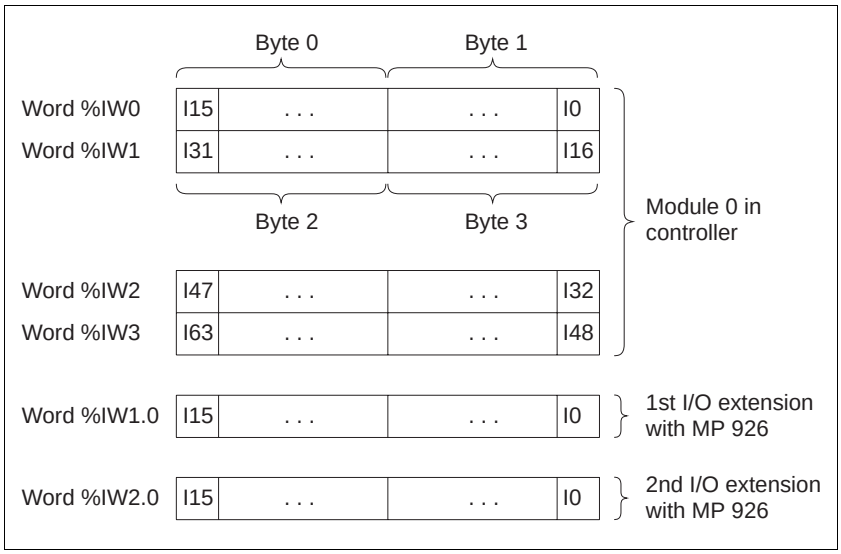


Fig. 1-17 Accessing inputs/outputs

ATTENTION
Byte access to @I and @Q as well as @QB and @IB is not admissible.

ATTENTION
If a module has 4 input words, the bit sequence is from @%IX <module number> 0 to @%IX <module number> 63.

NOTE
For information on input and output addressing, refer to the BPRO3 Operating Manual, chapter 3.4.6.

1.4.3 Variable and function block declaration

Local variables, input/output variables and function blocks must be declared in the block header of the respective block with the block header editor prior to their use. (Declaring means assigning a name and allocating memory space.) Variable and function block declarations always start with the string VAR... and end with the string VAR_END.

The following table shows the variable declarations and the block types in which they can be used:

Variable declaration	Meaning	Used in block type
VAR	Local variable or function block: Variable or function block is only recognized within the block.	PRG, FB, GLB, FUN
VAR_INPUT	Input variable: Variable for value input is read-only within the block.	FB, GLB, FUN
VAR_OUTPUT	Output variable: Variable for value output is write-only within the block.	FB, GLB
VAR_IN_OUT	Address for input/output variable: Variable for value input/output can be read and written.  *NOTE* *An input/output variable may be assigned only one variable when the block is called.*	FB, GLB, FUN
VAR_EXTERNAL	Declare variable from the assignment list to the block: Variable can be used within the block.	PRG, FB, GLB, FUN
VAR_RETAIN VAR_INPUT_RETAIN VAR_OUTPUT-RETAIN	Retentive variable: Value of variable is retained after power-off (meaning is analogous with VAR, VAR_INPUT, VAR_OUTPUT).	See VAR, VAR_INPUT, VAR_OUTPUT

Declaration examples:

```
VAR
   status 1      BOOL                  0      Declaration of local
                                              variables
   value 1       INT                   10
   number        ARRAY [1..9] OF INT
VAR_END

VAR_INPUT
   status 2      BOOL                  0      Declaration of input
                                              variables
   status 3      BOOL                  1
VAR_END

VAR
   tim_on_del    ton                          Declaration of a
                                              function block (of type
                                              ton)
VAR_END
```

1.4.4 Data block declaration

All variables declared in data blocks are global variables. A data block describes a data structure which can contain elementary and derived data types. The first step is to define the desired data structure of a data block using the “DB type editor”. You can then declare a data block of a specific data block type using the “DB editor”.

NOTE
Global blocks and data blocks do not have to be declared in the block header of the calling block.

```
Data block        :  poslist
Data block type   :  siglist
Author            :  J. Greene
Access level      :  8
Comment title     :  Position list block
 TYPE
 siglist                :  STRUCTURE
   pos0                    DINT              0
   pos1                    DINT              1000
   pos2                    DINT              2000
   pos3                    DINT              3000
   pos4                    DINT              4000
   pos5                    DINT              5000
   pos6                    DINT              6000
   END_STRUCTURE
 END_TYP
```

Fig. 1-18 Data block example

1.4.5 Variable initialization

When declaring a variable, it can be assigned a value (initialization). All variables are initialized when downloaded. In addition:

- Local variables of functions are initialized when calling the function.
- All variables, except retentive variables, are initialized when the pertaining task is invoked.
- All flags, except retentive flags, are initialized when the INIT task is invoked.

1.5 Symbolic names

Symbolic names improve the readability of a program. The following program elements can be identified with symbolic names:

– Variables
– Labels
– Blocks

A symbolic name for a variable can have a maximum length of 16 alphanumeric characters (A, a, B, b, ..., 0, 1, 2, 3, ...) and include an underline character (_). Special characters (%, &, #, ...) are not allowed. Names must have initial capitals and they are case-sensitive.

Valid names are	for example:
INP_1	Light_ON

Invalid names are, for example:

12_Output	(Name starts with a number)
Status#1	(Name includes a special character)

1.5.1 Variable names

Variables can have the following symbolic names:

Variable type	**Identifier**
Signal inputs	I or symbolic name according to assignment list
Signal outputs	Q or symbolic name according to assignment list
Flag	M or symbolic name according to assignment list
Other variable	Symbolic name according to declaration (see "Variable declaration")

Inputs/outputs and flags can be assigned symbolic names in the assignment list. However, they must be declared as VAR_EXTERNAL in the corresponding block.

NOTE
If the same name is used in a block for a local variable and for an input/output or flag in the assignment list, the corresponding input/output or flag cannot be referenced by this name in this block.

1.5.2 System constants

For parameter setting in control functions (see chapter on controller library), certain system constants are required. (e.g. axis number = “x1”, serial interface = “c”, encoder input 1 = “p1”). These names may not be used for any other purpose (e.g. variable, block name).

NOTE
The system constants are automatically inserted in the assignment list, and they do not have to be declared as VAR_EXTERNAL. For a table of the system constants, see chapter 8.

2 Programming in IL

2.1 General information on IL

An IL program or an IL block consists of a sequence of text instructions (IL = instruction list). An instruction in IL has the following format:

Operator (possibly with a **modifier**) and an **operand**

Example:

Operator	Operand	
ld	100	Load value 100 into CR

Current result (CR)

The current result (CR) is a temporary storage feature (accumulator) of the controller which is used for:

– Arithmetic operations
– Logical operations
– Passing data

IL example		**Comment**
ld	15	Load value 15 into CR
add	10	Add 10 to CR
st	Sum	Store contents of CR in the Sum variable

NOTE
In contrast to conventional programmable logic controllers, BERGER LAHR Series 300 controllers only have one accumulator for the CR. The memory size of the accumulator is automatically adjusted to the data type of the operand.

2.1.1 IL operators

The operator determines the operation to be performed with the operand and the CR. The following operators are defined:

Transfer operations

ld(n)	Load a value into the CR
st(n)	Store the CR in a variable

Setting and resetting

s	Set boolean operand to TRUE (“1”) if CR = TRUE
r	Set boolean operand to FALSE (“0”) if CR = TRUE

Logical operations

and(n)	AND operation
or(n)	OR operation
xor(n)	XOR operation

Arithmetic operations

add	Addition
sub	Subtraction
mul	Multiplication
div	Division

Relational operations

eq	equal to
le	less than or equal to
lt	less than
ge	greater than or equal to
gt	greater than

Block calls and jump operations

jmp(c, n)	Jump to a label
cal(c, n)	Function block call
ret(c, n)	Return from a block

Function calls used as operators

Operator field

Function names are used as operators since the function name is entered into the operator field. The function name specifies the operation to be performed with the CR ad any optional input variables. The result of a function is always stored in the CR.

2.1.2 Modifiers

Certain operators can take a modifier “c”, “n” or “(” which modifies the execution of the operation as described below.

Boolean negation

The modifier “n” can be used in a transfer operation or a logical operation to carry out a boolean negation with the operand before performing the operation itself. Negation is only possible with the BOOL, BYTE or WORD data types (for data types, see chapter 1.4.2).

IL example		Comment	
ldn	16#00	CR ← not 16#00	CR =16#FF
andn	16#ff	CR ← CR and (not 16#FF)	

Conditional jump operations

The modifiers “c” and “n” control the execution of the jump instructions “jmp”, “cal” and “ret”.
Jump instructions with the “c” modifier are only executed if the value in the CR is a boolean “1”, or TRUE.
Jump instructions with the “n” modifier are only executed if the value in the CR is a boolean “0”, or FALSE.

IL example		Comment
ld	value	CR ← value
eq	100	CR ← CR = 100
jmpc	label1	Jump if CR = TRUE

Parentheses

The modifier “(” can be used to modify the usual sequence of execution for logical, arithmetic and relational operations.
An opening parenthesis “(” must be used with the first operator, a closing parenthesis “)” must be entered in a subsequent operator field. A maximum of 10 nested parentheses are permitted.
Operations in parentheses are carried out before the operation in front of the parentheses.

IL example		Comment
ld	0	CR ← 0
and (	%IX0.1	CR ← 0 and (%IX0.1 OR %IX0.2)
or	%IX0.2	
)		

In this example, the current result is obtained as follows:
At first, an OR operation is carried out with the two input signals, then the AND operation is performed.

NOTE
An operator can take only one modifier.
For example, “andn (” is invalid.

2.1.3 Labels

Labels (the targets for the jump instructions “jmp”, “jmpc”, “jmpn”) must be placed at the start of IL networks. Example:

```
Network #1
    Label1:
<other IL networks or IL instructions>

    jmp          Label10

Network #10
    Label 10:
```

2.1.4 Operands

All IL instructions (except “ret”) require an operand with which to perform the operation. The following types of operands can be used in an instruction:

IL example		**Type of operand**
ld add ld	100 16#1000 12.34	Constant values
ld and	%MX1.14 %MX5.3	Flags
ld st ld st	%IX0.11 %QX0.3 @IB0 @QB1	Inputs/outputs, indirectly (from process image) Inputs/outputs, directly (from signal interface)
ld st ld st ld	value sum sum fbl1.input fbl1.output	Variables Input variable of a global or function block Output variable of a global or function block
cal	fbl1 (input:=10)	Function block or global block name with input/output variables
ld st ld st	array1 array2 array(5) array2(5)	Array as a whole Array element
ld st ld ld	dbname1 dbname2 dbname1.element1 dbname1.element2	Data block name Data block elements
jmp	label	Label

NOTE
The valid data types for operands are described in chapter 1.4.2. Some of the instructions can take several operands in an operand list with comma delimiters, e.g. add 100, 50, 200 (for the meaning, see the corresponding operation description in chapter 2.2).

2.1.5 Data type conversion

Before an IL operation is carried out, compatibility of the data types of CR and operator must be ensured. The following data type groups are compatible:

– INT and DINT,
– BOOL, BYTE and WORD

Automatic data type conversion

For operations with different but compatible data types (which belong to the same group), the CR is automatically converted to the more comprehensive data type.
The following should be noted for automatic data type conversion:

Automatic data type conversion	**Observations**
BOOL → BYTE	Lower bit of BYTE = BOOL, higher bits = 0
BOOL → WORD	Lower bit of WORD = BOOL, higher bits = 0
BYTE → WORD	Lower byte of WORD = BYTE, higher byte = 0
INT → DINT	DINT = INT

Example:

```
ld      2#01001001      Automatic data type conversion from BOOL
and     TRUE            to BYTE; the result is 2#00000001.
```

Data type conversion functions

If the CR and the operator contain incompatible data types, a data type conversion function must be called before the operation is carried out. The following data type conversion functions are included in the standard library of the programming system:

– dint_to_lreal
– lreal_to_dint
– dint_to_word
– word_to_dint
– time_to_lreal
– dint_to_time
– time_to_dint

Example:

```
ld          100         Data type conversion from INT
dint_to_lreal           to LREAL through function call
add         1.2
```

2.2 IL operation description

2.2.1 Transfer operations

The “ld” and “st” transfer operations can be used for loading values into the CR and for writing values from the CR to variables.

Operator	Modifier	Operand type	Comment
ld	n	BOOL, BYTE, WORD, INT, DINT, LREAL, TIME, STRING, ARRAY, DB	Load a value into the CR
st	n	BOOL, BYTE, WORD, INT, DINT, LREAL, TIME, STRING, ARRAY, DB	Store the CR in a variable

Modifiers

Boolean negation

The “n” modifier can be used for performing a boolean negation with the operand if it is of data type BOOL, BYTE, or WORD.

Data types

When loading (“ld”), the CR takes the same data type as the value loaded. When storing (“st”), the data types of the CR and the operand must be compatible (see chapter 2.1.5). Note the following rules when storing from → to:

Automatic data type conversion	Observations
BOOL → BYTE	Lower bit of BYTE = BOOL, higher bits = 0
BOOL → WORD	Lower bit of WORD = BOOL, higher bits = 0
BYTE → WORD	Lower byte of WORD = BYTE, higher byte = 0
WORD → BYTE	BYTE = Lower byte of WORD
BYTE → BOOL	If BYTE is not equal to 0, BOOL = 1
WORD → BOOL	If WORD is not equal to 0, BOOL = 1
INT → DINT	DINT = INT
DINT → INT	INT = DINT If the DINT number exceeds the range of values of INT, the INT number is set to the maximum value (+32767 or -32768).

The data type of the CR does not change when storing with “st”.

NOTE
For the derived data types STRING, ARRAY and DB, only the full data type can be stored after loading.

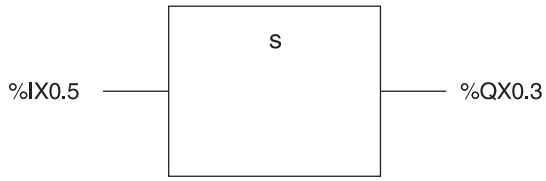
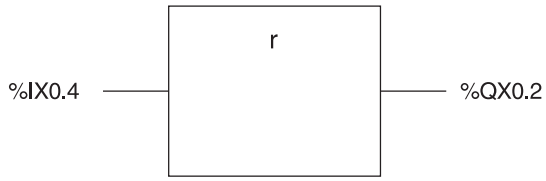
Examples:

```
VAR
    bytevariable            BYTE
    wordvariable            WORD
    realvariable            LREAL
VAR_END

ld          %MW20       CR=Word information
st          bytevariable    This word information
st          wordvariable    is retained in the CR
                            after storing.
ldn         bytevariable
st          wordvariable

ld          1.0
st          value
```

2.2.2 Setting and resetting

The set operation “s” and reset operation “r” can be used for setting a variable of type BOOL (e.g. output or flag) to “1” or resetting it to “0”, depending on the current result.

Operator	Modifier	Operand type	Comment
s		BOOL	Set boolean operand to “1” if the CR = 1 (if CR = 0, the status remains unchanged).
r		BOOL	Reset boolean operand to “0” if the CR = 1 (if CR = 0, the status remains unchanged).

IL example	FBD representation	Comment
ld %IX0.5 s %QX0.3		The output %QX0.3 is set to “1”, if CR = 1.
ld r %IX0.4 %QX0.2		The output %QX0.2 is set to “0”, if CR = 1.

2.2.3 Logical operations

The operators “and”, “or” and “xor” can be used to perform logical operations with one or more boolean operands and the CR.

Operator	Modifier	Operand type	Comment
and	n, (	BOOL, BYTE, WORD	AND operation
or	n, (	BOOL, BYTE, WORD	OR operation
xor	n, (	BOOL, BYTE, WORD	XOR operation

Modifiers

Boolean negation

The “n” modifier can be used before the actual operation for subjecting the operand to a boolean negation. The “n” modifier must follow the operator, e.g. orn.

Parentheses

The “(” modifier can be used for placing several logical operations into parentheses. For this purpose, an opening parenthesis “(” must be used in the operand field, a closing parenthesis “)” must be entered in a subsequent operator field; see chapter 2.1.2.

Data types

For logical operations with differing but compatible data types, an automatic data type conversion is carried out before the operation (see chapter 2.1.5). The following rules should be noted:

Automatic data type conversion	Observations
BOOL → BYTE	Lower bit of BYTE = BOOL, higher bits = 0
BOOL → WORD	Lower bit of WORD = BOOL, higher bits = 0
BYTE → WORD	Lower byte of WORD = BYTE, higher byte = 0

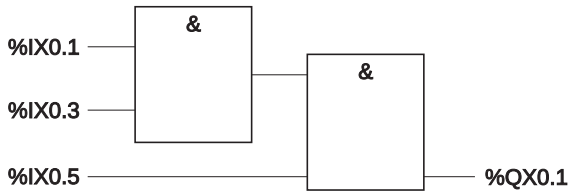
Operand list

NOTE
An operator can take only one modifier.
For example, “andn (%IX0.1 or %IX0.2)” is invalid.

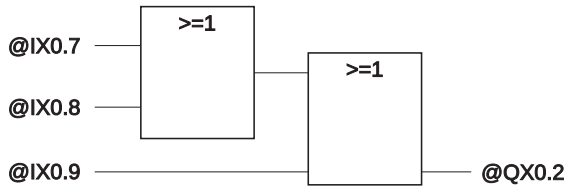
Several operands (up to a maximum of 9) can be entered using a comma delimiter. In this case, the result is determined as follows:

and	A, B, C, ...	CR	←	CR and A and B and C and ...
or	A, B, C, ...	CR	←	CR or A or B or C or ...
xor	A, B, C, ...	CR	←	CR xor A xor B xor C xor ...

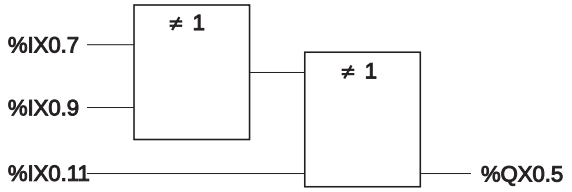
AND operation

IL example	FBD representation	Comment
ld %IX0.1 and %IX0.3 and %IX0.5 st %QX0.1		The output %QX0.1 has signal status “1” if the signal “1” is present on all inputs. Output %QX0.1 has signal status “0” as soon as signal “0” is present on any input. The operation instructions can be specified in any order.

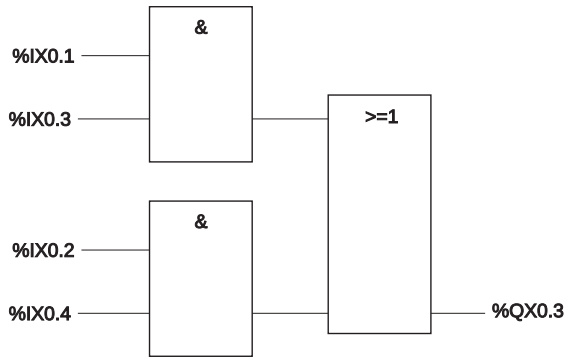
OR operation

IL example	FBD representation	Comment
ld @IX0.7 or @IX0.8 or @IX0.9 st @QX0.2		The output @QX0.2 has signal status “1” if the signal “1” is present on at least one input. Output @QX0.2 has signal status “0” if all inputs have signal status “0” simultaneously. The operation instructions can be specified in any order.

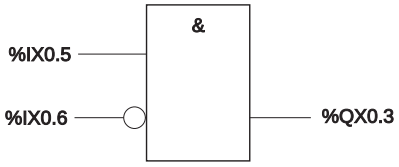
XOR operation

IL example	FBD representation	Comment
ld %IX0.7 xor %IX0.9 xor %IX0.11 st %QX0.5		The output %QX0.5 has signal status “1” if an odd number of “1”s is present on the inputs.

AND/OR operation

IL example	FBD representation	Comment
ld %IX0.1 and %IX0.3 or(%IX0.2 and %IX0.4) st %QX0.3		The output %QX0.3 has signal status “1” if an AND operation is true. Output %QX0.3 has signal status “0” if no AND operation is true.

Negated AND operation

IL example	FBD representation	Comment
ld @IX0.5 andn @IX0.6 @QX0.3 st		The output @QX0.3 has signal status “1” if signal status “1” is present on input @IX0.5 and input @IX0.6 has signal status “0”.

2.2.4 Arithmetic operations

The operators “add”, “sub”, “mul”, “div” can be used to carry out arithmetic operations between one or (for “add” and “mul”) several operands and the CR.

Operator	Modifier	Operand type	Comment
add	(	INT, DINT, LREAL, TIME	Addition, +
sub	(	INT, DINT, LREAL, TIME	Subtraction, -
mul	(	INT, DINT, LREAL, TIME	Multiplication, x
div	(	INT, DINT, LREAL, TIME	Division, /

Modifiers

Parentheses

The “(” modifier can be used for placing several arithmetic operations between parentheses. For this purpose, an opening parenthesis “(” must be used in the operand field, a closing parenthesis “)” must be entered in a subsequent operator field; see chapter 2.1.2.

Data types

For arithmetic operations with differing but compatible data types, an automatic data type conversion is carried out before the operation (see chapter 2.1.5). The following arithmetic operations are valid:

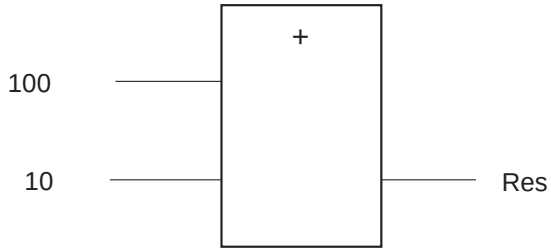
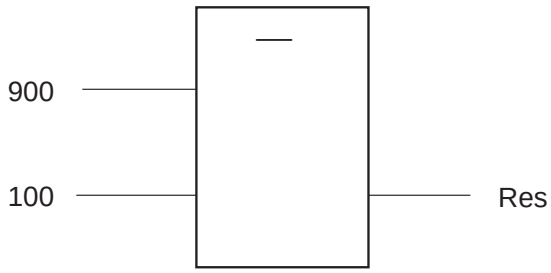
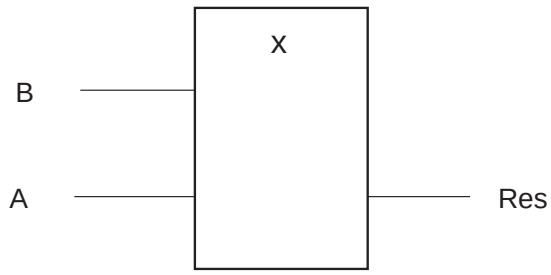
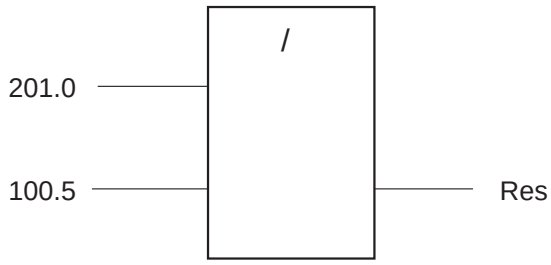
CR	Operation	Operand	→	Result (CR)
INT	+ , - , *, /	INT	→	INT
INT	+ , - , *, /	DINT	→	DINT
DINT	+ , - , *, /	INT	→	DINT
LREAL	+ , - , *, /	LREAL	→	LREAL
TIME	+ , -	TIME	→	TIME
TIME	* , /	INT	→	TIME
INT	* , /	TIME	→	TIME

Operand list

For the operations “add” and “mul”, several operands (up to a maximum of 9) can be entered using a comma delimiter. In this case, the result is determined as follows:

add A, B, C, ... CR ← CR + A + B + C + ...
mul A, B, C, ... CR ← CR x A x B x C x ...

The following table contains examples of arithmetic operations.

IL example	FBD representation	Comment
ld 100 add 10 st Res		The value 10 is added to the value of the CR (100). The result is stored in the “Res” variable.
ld 900 sub 100 st Res		The value 100 is subtracted from the value of the CR (900). The result is stored in the “Res” variable.
ld B mul A st Res		The value of the CR (variable B) is multiplied with the variable A. The result is stored in the “Res” variable.
ld 201.0 div 100.5 st Res		The value of the CR (value 201.0) is divided by the value 100.5. The result is stored in the “Res” variable.

2.2.5 Relational operations

The operators “eq”, “gt”, “ge”, “lt” and “le” can be used for relational operations between one or more operands and the CR. After the operation, the CR has always the data type BOOL and contains any of the following values:

– “0” or FALSE, if the result is not true or
– “1” or TRUE, if the result is true.

After a relational operation, a conditional jump operation may follow; see chapter 2.1.5.

Operator	Modifier	Operand type	Comment
eq	(	BOOL, BYTE, WORD, INT, DINT, LREAL, TIME	Equal to, =
gt	(	BOOL, BYTE, WORD, INT, DINT, LREAL, TIME	Greater than, >
ge	(	BOOL, BYTE, WORD, INT, DINT, LREAL, TIME	Greater than or equal to, >=
lt	(	BOOL, BYTE, WORD, INT, DINT, LREAL, TIME	Less than, <
le	(	BOOL, BYTE, WORD, INT, DINT, LREAL, TIME	Less than or equal to, <=

Modifiers

Parentheses

The “(” modifier can be used for placing several relational operations between parentheses. For this purpose, an opening parenthesis “(” must be used in the operand field, a closing parenthesis “)” must be entered in a subsequent operator field; see chapter 2.1.2.

Data types

For relational operations with differing but compatible data types, an automatic data type conversion is carried out before the operation (see chapter 2.1.5). The following rules should be noted:

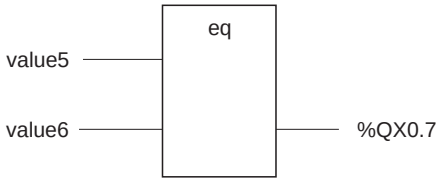
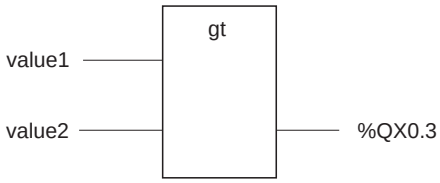
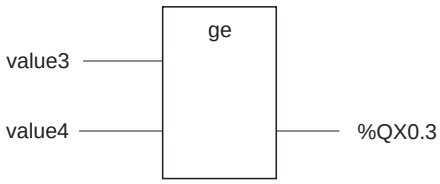
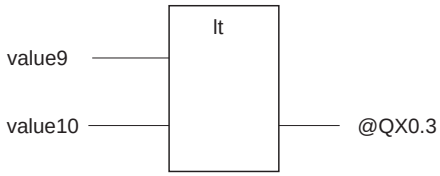
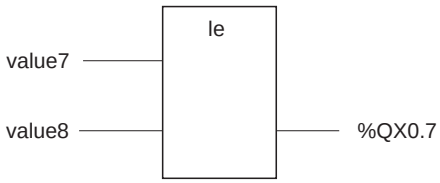
Automatic data type conversion	**Observations**
BOOL → BYTE	Lower bit of BYTE = BOOL, higher bits = 0
BOOL → WORD	Lower bit of WORD = BOOL, higher bits = 0
BYTE → WORD	Lower byte of WORD = BYTE, higher byte = 0
INT → DINT	DINT = INT

Operand list

The following relational operations can take several operands (up to a maximum of 9), delimited by commas. In this case, the result is determined as follows:

```
eq   A, B, C, ...   CR   ←   (CR = A) & (A = B) & (B = C) & ...
gt   A, B, C, ...   CR   ←   (CR > A) & (A > B) & (B > C) & ...
ge   A, B, C, ...   CR   ←   (CR >= A) & (A >= B) & (B >= C) & ...
lt   A, B, C, ...   CR   ←   (CR < A) & (A < B) & (B < C) & ...
le   A, B, C, ...   CR   ←   (CR <= A) & (A <= B) & (B <= C) & ...
```

Relational operation examples

IL	FBD	Example
ld value5 eq value6 st %QX0.7		Comparison between the values **value5 = value6** The first value is compared to the second one according to the specified relational operation. If the comparison is true (match), the CR = 1, otherwise it is “0”.
ld value1 gt value2 st %QX0.3		Comparison between the values **value1 > value2** The first value is compared to the second one according to the specified relational operation. If the comparison is true (match), the CR = 1, otherwise it is “0”.
ld value3 ge value4 st %QX0.3		Comparison between the values **value3 > value4** The first value is compared to the second one according to the specified relational operation. If the comparison is true (match), the CR = 1, otherwise it is “0”.
ld value9 lt value10 st @QX0.3		Comparison between the values **value9 > value10** The first value is compared to the second one according to the specified relational operation. If the comparison is true (match), the CR = 1, otherwise it is “0”.
ld value7 le value8 st %QX0.7		Comparison between the values **value7 > value8** The first value is compared to the second one according to the specified relational operation. If the comparison is true (match), the CR = 1, otherwise it is “0”.

2.2.6 Block calls and jump operations

The operators “jmp”, “cal” and “ret” can be used for conditional or unconditional jump operations or block calls.

ATTENTION
For a conditional jump to a global or function block, note that the specified parameters are always passed to the block (whether a jump is executed or not).

Operator	Modifier	Operand type	Comment
jmp	c, n	Label	Jump to a specified label
cal	c, n	Name of a global or function block (with optional parameter list)	Call a global or function block
ret	c, n	No operand	Return from function, global or function block

Modifiers

The “c” and “n” modifiers can be used for conditional jump operations or block calls.
Jump instructions with the “c” modifier are only executed if the value in the CR has data type BOOL, BYTE or WORD and is greater than “0”.
Jump instructions with the “n” modifier are only executed if the value in the CR has data type BOOL, BYTE or WORD and is equal to “0”.

Operands

– Labels

  The operand field of the jump instruction “jmp” must include the name (up to 16 alphanumeric characters) of the target label. Labels can only be entered at the start of an IL network.

– Block names

  When calling global or function blocks with “cal”, the name of the global or function block to be called must be specified in the operand field. The name can be followed by an input parameter list in parentheses.

NOTE
Function blocks must be declared in the block header of the calling block; see chapter 1.4.5.

Block calls

There are three different methods of calling a global or function block. This is illustrated below using the standard function block “ctu” (incremental counter):
Declaring the function block in the block header of the calling block:

```
VAR
   Counter10   ctu    Function block Counter10 of type ctu
                      (incremental counter from block library)
VAR_END
```

1. Invoking “cal” with a parameter list

IL example		**Comment**
cal	Counter10 (cu:=%IX0.10, pv:=15)	Counter10 with a parameter list

When calling a function block with a parameter list, the input parameter values are loaded into the corresponding data structure. Parameters not specified in the parameter list retain their previous values.

2. Parameter passing with “ld/st” and subsequent call to “cal”

IL example		**Comment**
ld	15	Load value 15 into CR
st	Counter10.pv	Parameter passing
ld	%IX0.10	Load input 10
st	Counter10.cu	Parameter passing
cal	Counter10	Call

When calling a function block with combined load and store operations, the input parameter values are loaded into the corresponding data structure. Those values which are not assigned new values by “ld” and/or “st” retain their previous values.

3. Calling with an input parameter as the operator

IL example		**Comment**
ld	15	Load value 15 into CR
pv	Counter10	Parameter passing and call
ld	%IX0.10	Load input 10
cu	Counter10	Parameter passing and call

When calling a function block with a parameter in the operator field, the contents of the CR is copied to the same parameter in the block header. Subsequently, the block body is processed. Any further parameters must be loaded previously with an “ld” or “st” combination.

NOTE
The variables of global or function blocks retain their values until the block is called again.

Jump operations

Conditional jump operations

IL example	Comment
Network 1 ld value eq 100 jmpc Label : Network 3 Label:	 Load value into CR Compare value to 100 Jump if value in CR is equal to 100

IL example	Comment
Network 1 ld value eq 100 jmpn Label : Network 3 Label:	 Load value into CR Compare value to 100 Jump if value in CR is not equal to 100

Unconditional jump operation

IL example	Comment
Network 1 : jmp Label : Network 3 Label:	 Jump unconditionally, regardless of the value in the CR

3 Movement programming

Axis programming

For axis programming, a number of predefined functions are available in the controller library of the programming system (see block library).

The following section explains concepts and relationships involved in movement programming.

3.1 Axis operating modes

The “setmode” function can be used to select the following operating modes for controlling the axis (the “getmode” function can be used to retrieve the current operating mode.

NOTE
The axis operating mode can be changed whenever the axis is at a standstill.

NOTE
For the axis operating mode default, refer to the controller manual.

3.1.1 Point-to-point mode

Absolute positioning
Relative positioning

In point-to-point mode, a positioning command is used for moving from point A to point B. Positioning can be absolute (based on the axis zero position) or relative (based on the current axis position). In addition, you can specify whether the program should wait until the positioning operation is completed or continue immediately. Position values can be specified in user-defined units, e.g. 100 (mm), or drive units, e.g. 1000 (increments); see chapter 3.2.

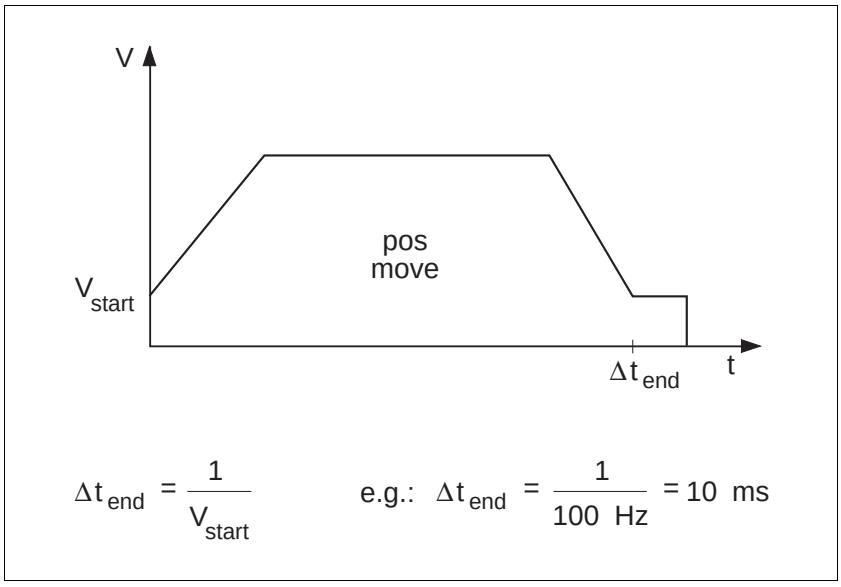


Fig. 3-1 Point-to-point mode

Point-to-point mode movement presettings:

Function	**Meaning**
accel	Select acceleration/deceleration curve
acclin	Calculate and activate linear acceleration/deceleration curve
calcaccel	Calculate acceleration/deceleration curve
setaccsig	Signal-dependent deceleration
setnorm	Set normalizing factors
setpos	Set positions in user-defined units
setposd	Set positions in drive units
setsiglist	Set/reset signal output depending on position
setvel	Set start and maximum speeds
setvellist	Change set speed depending on position
vel	Set the set speed

Functions to initiate or stop movements in point-to-point mode:

Function	**Meaning**
continue	Continue interrupted axis movement
move	Relative positioning operation in user-defined units
moved	Relative positioning operation in drive units
movedf	Relative positioning operation in drive units and wait until position reached
movedw	Relative positioning operation in drive units and wait for end of movement
movef	Relative positioning operation in user-defined units and wait until position reached
movew	Relative positioning operation in user-defined units and wait for end of movement
pos	Absolute positioning operation in user-defined units
posd	Absolute positioning operation in drive units
posdf	Absolute positioning operation in drive units and wait until position reached
posdw	Absolute positioning operation in drive units and wait for end of movement
posf	Absolute positioning operation in user-defined units and wait until position reached
posw	Absolute positioning operation in user-defined units and wait for end of movement
refpos	Reference movement
refposf	Reference movement and wait until reference point reached
refposw	Reference movement and wait for end of movement
stop	Stop axis movement

NOTE
During positioning with the "pos" function, the program continues to execute, i.e. input signals can be read, output signals can be set, the path or the speed for positioning can be changed.

NOTE
Positioning functions with wait states should be included in the SEQUENCE task; see chapter 1.2.4.3.

3.1.2 Speed mode

In speed mode, the “vel” function is used for setting a set speed and starting a movement. The axis continues to move at this speed until a different set speed is defined. The speed is specified in user-defined units, e.g. 10 (cm/s).

Speed mode movement presettings:

Function	**Meaning**
accel	Select acceleration/deceleration curve
acclin	Calculate and activate linear acceleration/deceleration curve
calcaccel	Calculate acceleration/deceleration curve
setaccsig	Signal-dependent deceleration
setnorm	Set normalizing factors
setpos	Set positions in user-defined units
setposd	Set positions in drive units
setsiglist	Set/reset signal output depending on position
setvel	Set maximum speeds
setvellist	Change set speed depending on position

Functions to initiate or stop movements in speed mode:

Function	**Meaning**
continue	Continue interrupted axis movement
stop	Stop axis movement
vel	Set the set speed and start axis movement

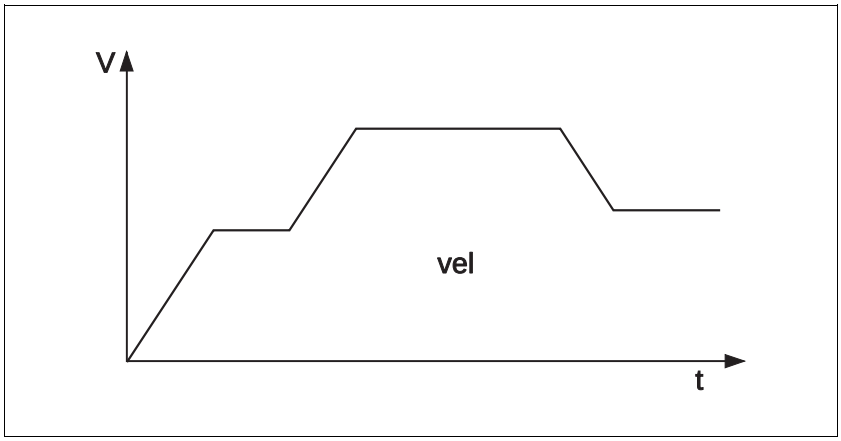


Fig. 3-2 Speed mode

3.1.3 Position following mode

In position following mode, positions are preset via an encoder or a variable. The setpoint is retrieved and the axis positioned using an adjustable time base (sampling time). The setpoint can be modified with a gear ratio (see chapter 3.2) so that an electronic gear can be implemented.
Position following mode movement presettings:

Function	Meaning
accel	Select acceleration/deceleration curve
acclin	Calculate and activate linear acceleration/deceleration curve
calcaccel	Calculate acceleration/deceleration curve
setnorm	Set gear ratio
setnormlist	Change gear ratios depending on position
setpos	Set positions and position offsets in user-defined units
setposd	Set positions and position offsets in drive units
setsiglist	Set/reset signal output depending on position
setsrcres	Set encoder input for position following mode
setsrctime	Set sampling time for position following mode
setsrctyp	Set reference variable type for position following mode
setsrcvar	Define variable for position following mode
setvel	Set maximum speeds
setvellist	Change set speed depending on position

Movements are controlled in position following mode by a pulse signal on the encoder input or by the value of a variable. The “setsrctyp” function can be used to specify the reference variable type:

– Encoder input

  The frequency of the pulse signal on the encoder input selected with “setsrcres” determines the acceleration and the speed of the axis. If no pulses are present, this is equivalent to axis standstill. The following system limit values must be observed: maximum system speed and maximum acceleration of the selected acceleration/deceleration curve.

– Variable

  The value of a variable defined with “setsrcvar” is interpreted as the absolute position for the axis. When this value changes, the axis moves to the new position using the selected acceleration curve and the set speed.

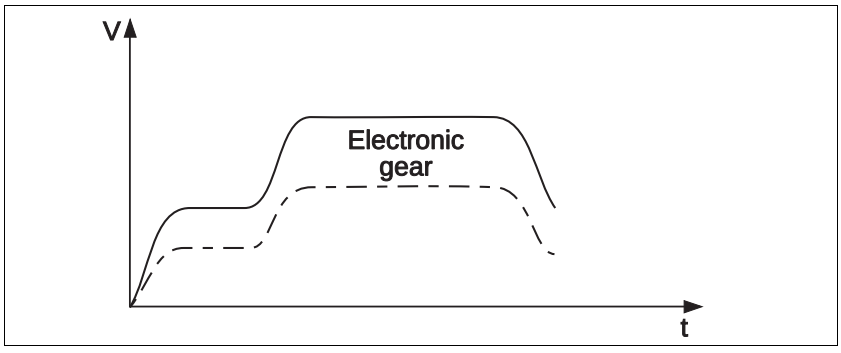


Fig. 3-3 Position following mode

3.2 Normalizing factors

The “setnorm” function can be used to define various normalizing factors. The normalizing factors are provided to:

– Convert user-defined units to drive units
– Synchronize the encoder with the axis
– Implement a step-up or step-down gearing with a reference variable in position following mode (electronic gear).

NOTE
A normalizing factor is always made up of a numerator and a denominator.

Drive units

Drive units are processing parameters internal to the controller. Drive units are defined as follows:

– Drive units for positions = Motor steps
– Drive units for speeds = Motor steps/s/256
– Drive units for accelerations = Motor steps/s^2

User-defined units

User-defined units are processing parameters which can be freely defined by the user. The following applies:

Drive units = User-defined units x Normalizing factor

Positions can be specified in drive units or in user-defined units.
Speeds and maximum accelerations (for calculating acceleration curves) can only be specified in user-defined units.

NOTE
Units of measurement:
When entering user-defined units, units of measurement such as mm, Hz, m/s can be used. This must be specified to the programming system from the menue option “Units of measure”. A number entered with a unit of measurement will be multiplied with the appropriate unit of measurement factor.

Encoder synchronization

If an encoder is used (e.g. for rotation monitoring), it must be synchronized with the shaft, i.e. the controller must be informed how many encoder units (increments) correspond to one drive unit (motor step). The following applies:

Encoder units = Drive units x Normalizing factor

Example:
For a stepping motor with 1000 steps per revolution and an encoder with 4000 encoder units per revolution, a normalizing factor of 4/1 must be specified.

The encoder unit (increment) is derived from the resolution (encoder marks per revolution) of the encoder used and from the internal evaluation factor. The following applies:

Encoder units = Encoder marks x Internal evaluation factor
(single, double, quadruple)

The evaluation factor is defined with the “setmode” function.

Electronic gear

In position following mode, a normalizing factor can be applied to the reference variable (e.g. encoder) to implement step-up or step-down gearing. This is also referred to as an electronic gear. The following applies:

Drive units = Reference variable x Normalizing factor

3.3 Acceleration curves, speeds and positions

3.3.1 Acceleration curves

The form of acceleration curves is freely programmable. For this purpose, a so-called master curve with relative speed/acceleration value pairs must be stored in a data block. The master curves must have the following tabular format:

Numerator for speed normalization	Denominator for speed normalization
Relative speed 1	Relative acceleration value 1
Relative speed 2	Relative acceleration value 2
Relative speed 3	Relative acceleration value 3
etc.	etc.
0 (last value)	Relative acceleration value n (n = 255 max.)

NOTE
The values in the table must have the data type DINT. It is recommended to select 1000 as the maximum relative acceleration value. The last speed value in the table must be 0.

Subsequently, the master curve is used with the “calcaccel” function to calculate the acceleration curve for the corresponding shaft load. For this purpose, the maximum acceleration must be specified with a number, which is to be used for storing the calculated acceleration curve (a maximum of 10 different acceleration curves can be stored).

Based on the maximum acceleration and the start speed preset with the “setvel” function, the acceleration values up to the maximum system speed are calculated.

```
Data block        :  V5913D800
Data block type   :  accel_curve
Author            :  J. Greene
Access level      :  8
Comment title     :  Master curve for VRDM5913 with WDP318
TYPE
accel_curve       :  STRUCTURE
  numerator          DINT          256
  denominator        DINT          1
  v0                 DINT          2000
  a0                 DINT          1000
  v1                 DINT          3000
  a1                 DINT          950
  v2                 DINT          5000
  a2                 DINT          900
  v3                 DINT          8000
  a3                 DINT          800
  v4                 DINT          10000
  a4                 DINT          600
  v5                 DINT          20000
  a5                 DINT          400
  v6                 DINT          30000
  a6                 DINT          300
  v7                 DINT          50000
  a7                 DINT          150
  v8                 DINT          100000
  a8                 DINT          50
  v9                 DINT          0
  a9                 DINT          10
  END_STRUCTURE
END_TYP
```

Bild 3-4 Master curve for VRDM 5913 motor

The highest relative acceleration value within the speed range corresponds to the maximum acceleration.

Example: with speed normalizing factor 256 (= Hz)

Master curve

256	1
2000	1000
3000	950
5000	900
8000	800
10000	600
20000	400
30000	300
50000	150
100000	50
0	10

→ calcaccel →	
Start speed	= 1000 (Hz)
Max. system speed	= 10000 (Hz)
Maximum acceleration	= 500 (Hz/ms)
Acceleration curve	= 1
→ calcaccel →	
Start speed	= 1000 (Hz)
Max. system speed	= 10000 (Hz)
Maximum acceleration	= 200 (Hz/ms)
Acceleration curve	= 2

Acceleration curve 1

2000 (Hz)	500 (Hz/ms)
3000 (Hz)	475 (Hz/ms)
5000 (Hz)	450 (Hz/ms)
8000 (Hz)	400 (Hz/ms)
10000 (Hz)	300 (Hz/ms)

Acceleration curve 2

2000 (Hz)	200 (Hz/ms)
3000 (Hz)	190 (Hz/ms)
5000 (Hz)	180 (Hz/ms)
8000 (Hz)	160 (Hz/ms)
10000 (Hz)	120 (Hz/ms)

The acceleration curve 1 is interpreted as follows during acceleration: up to a speed value of 2000 Hz, acceleration is 500 Hz/ms, from 2000 Hz to 3000 Hz, acceleration is 475 Hz/ms, etc.
The acceleration curve is calculated up to the maximum system speed. For deceleration, the table is interpreted inversely. The “accel” function can be used to select the acceleration curve to be used for acceleration or deceleration.
The “acclin” function can be used to calculate and activate a linear acceleration curve; see controller library.

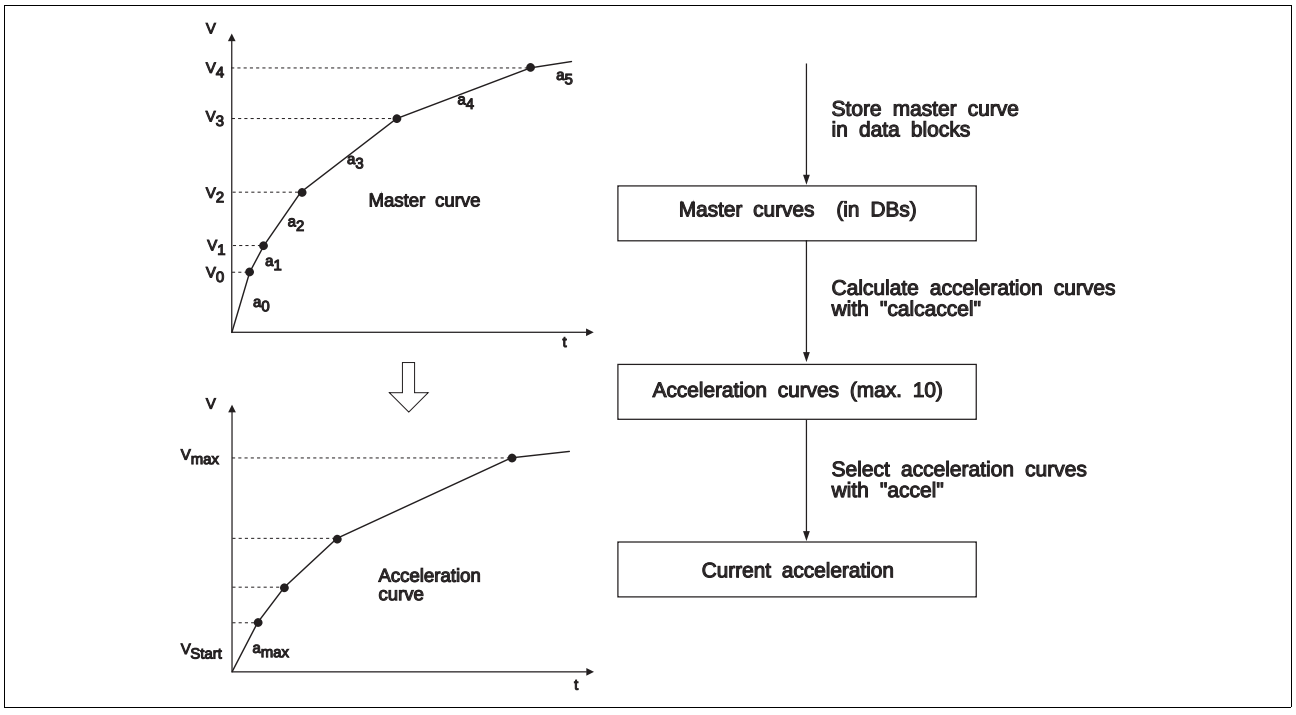


Fig. 3-5 Acceleraton curves

3.3.2 Signal-dependent deceleration (emergency deceleration ramps)

Certain input signals can be linked to acceleration curves for quick braking, e.g. for emergency stop or reference movement (see “setaccsig” function). As soon as the corresponding input signal is active, the shaft is decelerated using the linked acceleration curve.

3.3.3 Predefined signals

There are predefined external and internal axis and encoder signals for the controller which stop a movement when activated. For example:

- Hardware/software limit switches (“limp”, “limn”)
- Reference switch (“ref”)
- Hardware/software stop (“stop”)
- Power controller readiness (“ampnotready”)
- Contouring error (“dragerr”)

The “getsig” function can be used to directly retrieve the status conditions (deenergized = 0, energized = 1) of these signals. With “getsig_sr”, the temporarily stored active states of these signals can be retrieved.

The “setsig_activ_h” function can be used to set the active state (energized = active or deenergized = active) of the signals, and the “getsig_activ_h” function can be used for retrieving this state.

Signals can be reset in temporary storage and an interrupted movement resumed using the “clrsig_sr” function.

The “ensig” function can be used for enabling or disabling the signals. Only those signals that are enabled with “ensig” are enabled in the predefined way. Disabled external signals (e.g. reference switches) can be used freely. The “getensig” function can be used to determine whether a signal is enabled or disabled.

Interaction of the individual functions using the “limp”, “limn”, “ref” and “stop” signals.

NOTE
Any of these signals may interrupt a movement until disabling with the “ensig” function.
It is therefore necessary after the “ensig” function to cancel any movement interruption (drive is in blocked status) using the “clrsig_sr” function.

Example:

```
ld          watch
getensig    x1          Read enable signals
andn        limp
ensig       x1          Disable “limp” signal
ld          limp
clrsig_sr   x1          Cancel movement interruption by “limp”
```

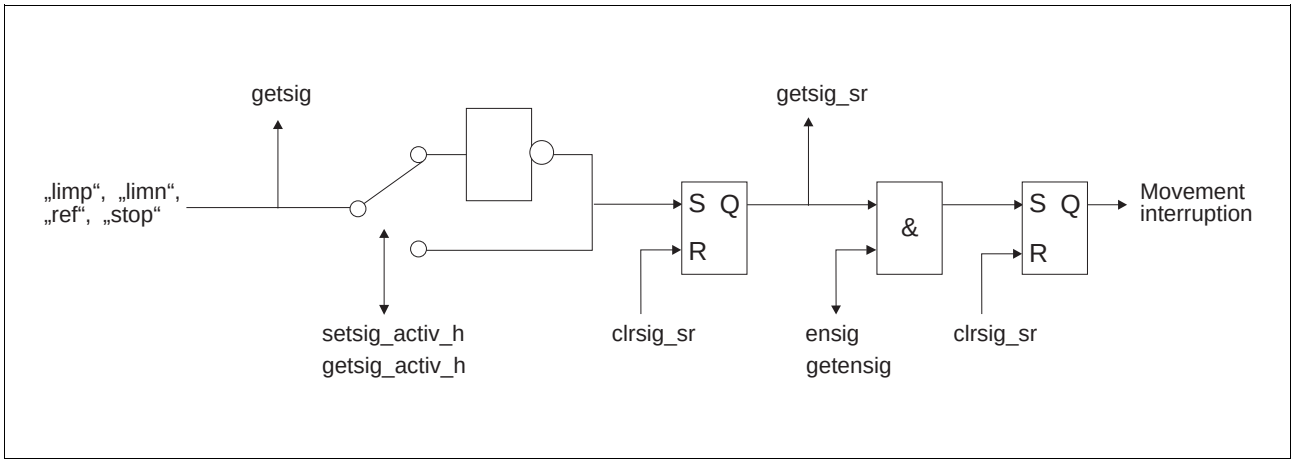


Fig. 3-6 Signals „limp“, “limn”, “ref” and “stop”

Interaction of the individual functions using the “ampready” signal.

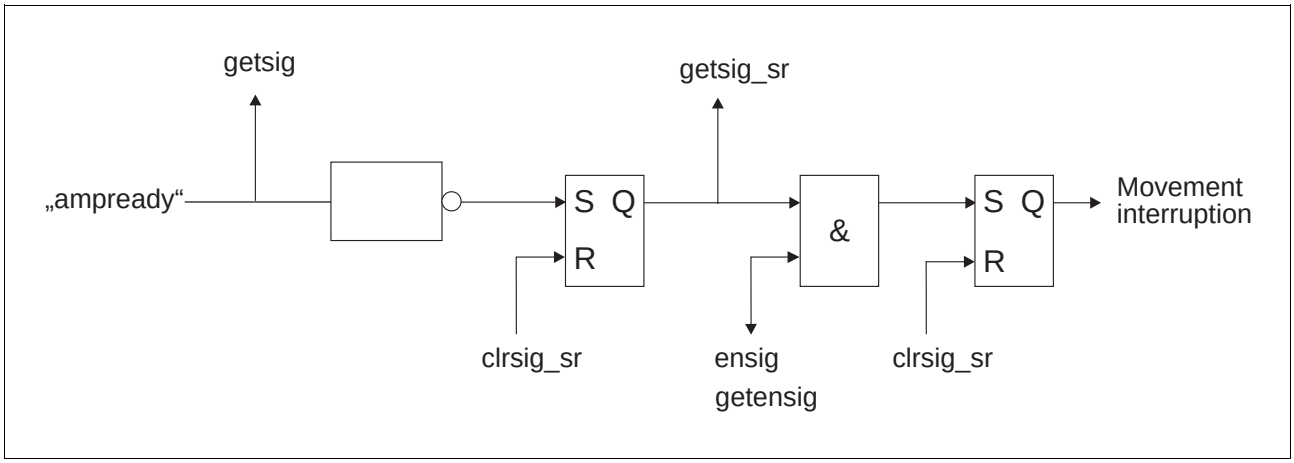


Fig. 3-7 Signal „ampready“

Interaction of the individual functions using the “dragerr” signal.

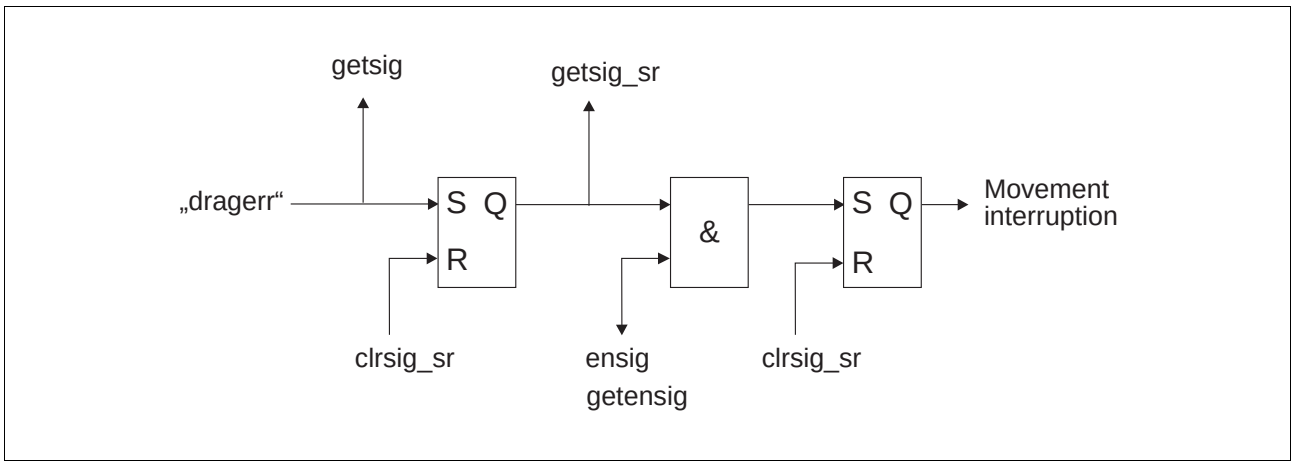


Fig. 3-8 Signal „dragerr“

ATTENTION
When using WDP3-337 or WDP3-338 units, the monitoring signals “ampnotready” and “dragerr” must not be disabled.

3.3.4 Speeds

Fixed speeds

The “setvel” function can be used to define the following fixed speeds:

– Start speed
– Maximum system speed

Set speeds

The “vel” function can be used for specifying set speeds. In speed mode, a shaft movement is initiated in addition.

3.3.5 Store actual position

Storing the actual position using the “trig” input or the encoder p2 pulse.

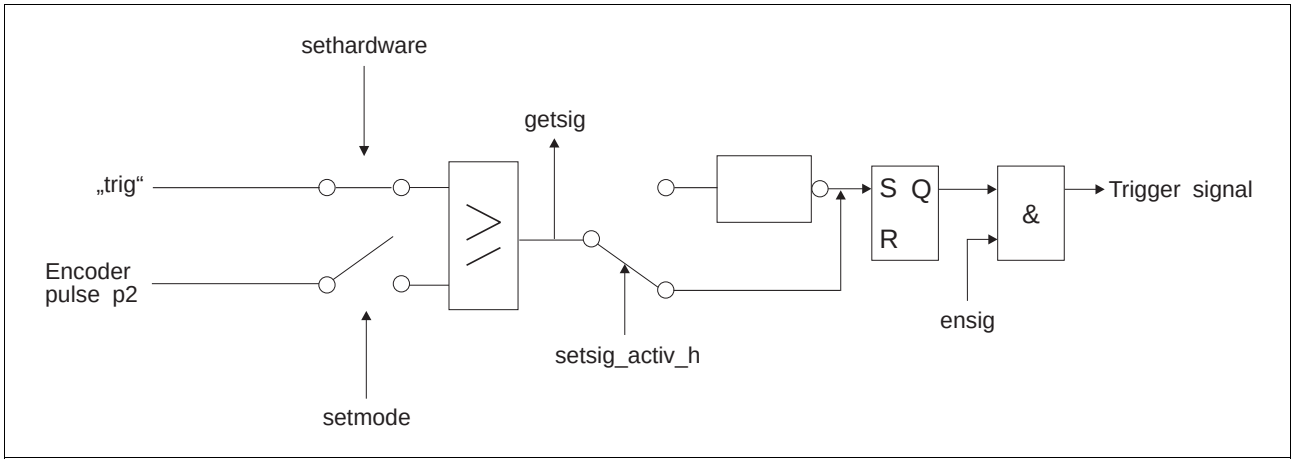


Fig. 3-9 Store the actual position

NOTE
The “clrsig_sr” function is used for preparing a new trigger event. If an active level is still present on the selected trigger input, this will cause immediate triggering. If the trigger function is to operate on an edge-selective basis, “clrsig_sr” must not be called until there is no more active level on the trigger input.

Example:

```
wait:

ld          watch
getsig      x1          Read input
and         trig
eq          trig        Trigger input still active?
jmpc        wait
ld          trig
clrsig_sr   x1          Reset trigger
```

3.3.6 Positions

Fixed positions

The “setpos” function can be used to set the following fixed positions:

– Current position (to set dimensions)
– Software limit switch position
– Position offset for position following mode
– Safety distance from limit or reference switch
– Encoder position
– Contouring error limit

Setpoint (set position)

Setpoints are predefined in point-to-point mode with the “pos” or “move” functions. At the same time, a positioning operation is initiated.

In position following mode, setpoints are preset through an encoder input or a variable.

Position lists

The “setsiglist”, “setvellist” and “setnormlist” functions can be used to activate position lists which are stored in data blocks in tabular format.

Position lists can be used to initiate the following position-related drive operations:

Function	Position-related drive operation	Possible axis operating modes
setsiglist	Set and reset a signal output (toggling)	All axis operating modes
setvellist	Change set speeds	Point-to-point mode, speed mode
setnormlist	Change gear ratios in position following mode	Position following mode

The position lists must have the following structure:

Position/signal list for “setsiglist”

Position 0
Position 1
.
.
.
Position n

Position/speed list for “setvellist”

Position 0	Speed 0
Position 1	Speed 1
.	.
.	.
.	.
Position n	Speed n

Position/gear ratio list for “setnormlist”

Position 0	Numerator 0	Denominator 0
Position 1	Numerator 1	Denominator 1
.	.	.
.	.	.
.	.	.
Position n	Numerator n	Denominator n

NOTE
The values in the tables must have the data type DINT. The positions are interpreted as absolute positions in drive units. Speeds are interpreted as user-defined units (with the current speed normalizing factor).

When a position list is activated it is processed once from top to bottom.

Example of a position/speed list:

```
Data block        :  flist
Data block type   :  frequlist
Author            :  J. Greene
Access level      :  8
Comment title     :  Example of a position/speed list
 TYPE
 frequlist        :  STRUCTURE
   pos0              DINT          1000
   v0                DINT          5000
   pos1              DINT          6000
   v1                DINT          20000
   pos2              DINT          15000
   v2                DINT          1000
   pos3              DINT          20000
   v3                DINT          30000
   pos4              DINT          50000
   v4                DINT          200
   pos5              DINT          55000
   v5                DINT          5000
   pos6              DINT          60000
   v6                DINT          20000
   pos7              DINT          70000
   v7                DINT          1000
   pos8              DINT          80000
   v8                DINT          30000
   pos9              DINT          90000
   v9                DINT          200
   END_STRUCTURE
 END_TYP
```

Fig. 3-10 Example of a position/speed list

After activation with “setvellist”, the speed is changed to 5000 upon passing position 1000. At position 6000, the set speed is changed to 20000, and so on.

NOTE
Processing of a position list can be stopped with the “list” function. Refer to the corresponding axis operating mode in chapter 3.1 for the functions which initiate movements.

3.4 Reference movement

The “refpos” “refposw” and “refposf” functions can be used for performing reference movements.

NOTE
Reference movements are only possible in point-to-point mode.

In a reference movement, a reference point is approached which is to be the zero point for the system of dimensions.

Reference movements can address the

– negative limit switch,
– positive limit switch or
– reference switch.

The principles of the different reference movements are illustrated in the figures 3-11 and 3-12.

The “getstate” function can be used for reading several reference movement status conditions.

The “setpos” function can be used for programming a safety distance to the limit switch or reference switch.

NOTE
Instead of performing a reference movement it is also possible to set a reference point for the system of dimensions using the “setpos” function.

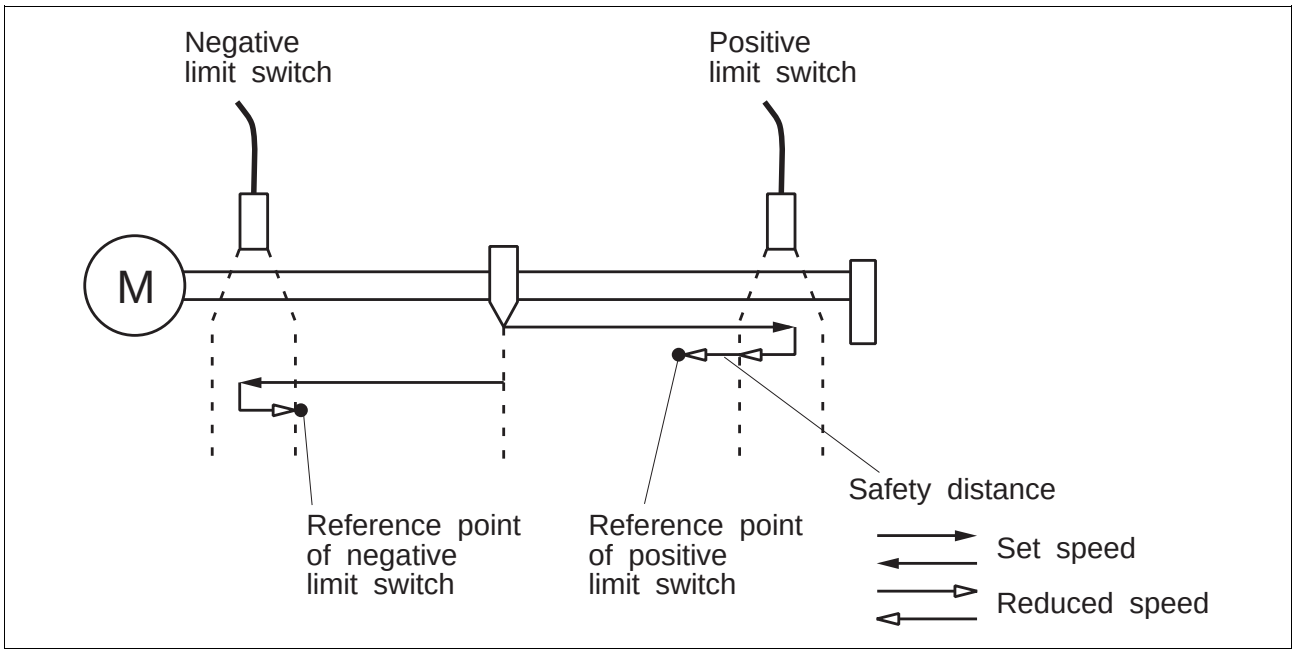


Fig. 3-11 Reference movement with limit switches

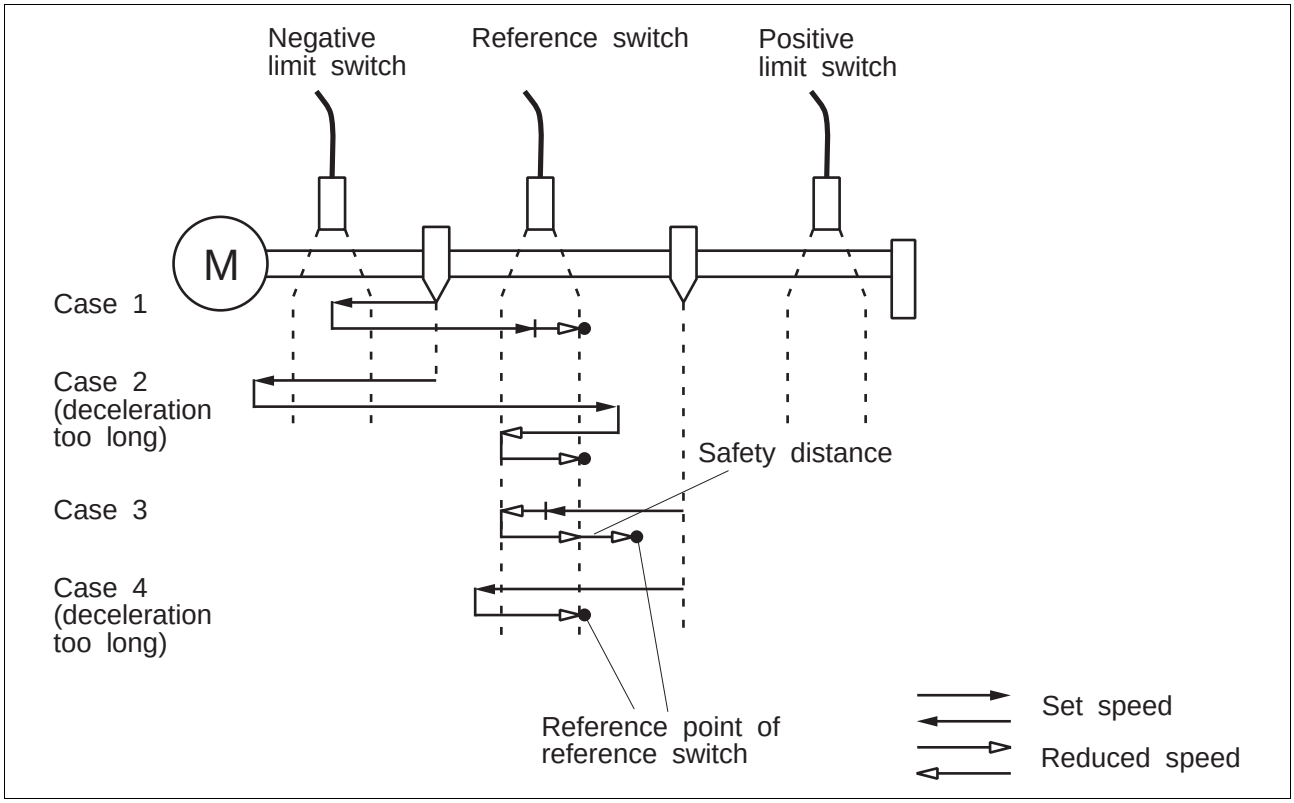


Fig. 3-12 Reference movement with limit switches

To determine the “reference movement completed” status, only use the “ref” option of the “getstate” function.

Example:

```
ld          ref
getstate    x1
and         refactiv
```

Never use the “action” or “motion” option for this read operation.

3.5 Encoder programming

Each encoder input can be used for

– reference variable input (in position following mode) or
– rotation monitoring (on WDP3-337 and WDP3-338 controllers by way of actual position monitoring).

NOTE
To facilitate programming of the encoder, the programming system (project: "Library") includes functions such as "watchaxis" for rotation monitoring and "encsrc" for encoder as the reference variable source.

Figure 3-13 illustrates the functions for reference variable input or rotation monitoring and their effects on the controller.

NOTE
If an encoder is used (e.g. for rotation monitoring), it must be synchronized with the shaft; see the section on encoder synchronization in chapter 3.2, "Normalizing factors".

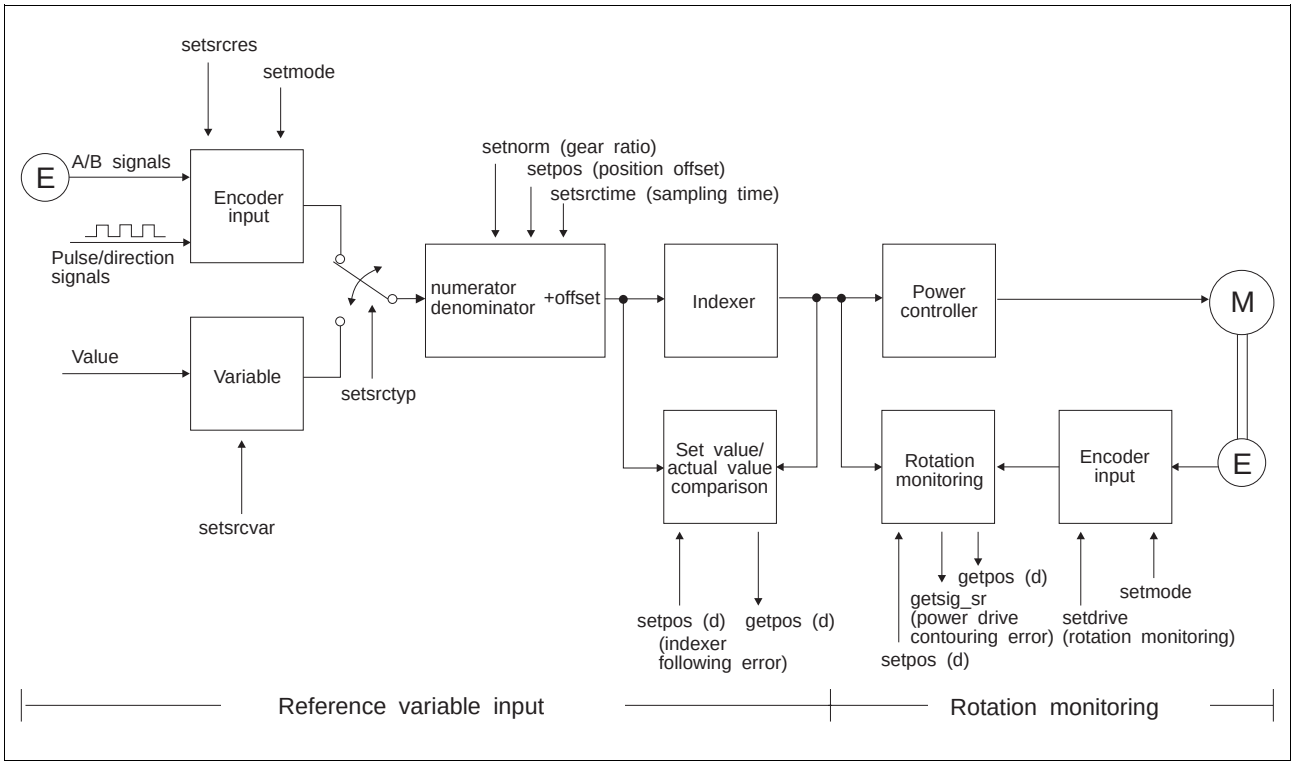


Fig. 3-13 Encoder functions

3.5.1 Reference variable input (e.g. for electronic gear)

The following controller library functions are relevant for reference variable input:

Function	Meaning
setmode	Set axis operating mode and encoder input
setsrctyp	Set reference variable source (variable or encoder)
setsrcres	Set encoder input for position following mode
setsrcvar	Set variable for position following mode
setnorm	Set normalizing factors (e.g. gear ratio)
setpos(d)	Input positions (e.g. position offset)
getpos(d)	Retrieve positions or indexer contouring error

In position following mode, the reference variable (position) is preset by

– an encoder input or
– a variable.

The reference variable is multiplied with a gear ratio. This can be used for implementing an electronic gear.
In addition, the “setpos” or “setposd” functions can be used for modifying the reference variable with a position value (position offset).

If the reference variable changes too quickly, the programmed acceleration curve is used to accelerate or decelerate. The “getpos” or “getposd” function can be used to determine the difference (following error) between the reference variable and the actual position of the indexer. A temporary storage space is provided to ensure that no pulses are lost.

NOTE
If the drive is stopped, or if a different axis operating mode is selected, the supplied pulses (positions) continue to be collected and temporarily stored. When returning to position following mode, these pulses are read and result in a relative positioning operation of the drive.

NOTE
All controllers with control versions up to 03.01 require the indexer contouring error to be be set to 0 before activating position following mode (see programming example).

Programming examples for position following mode

Encoder is reference source:

```
ld          srcenc
setsrctyp   x1
ld          p2
setsrcres   x1
ld          source
setnorm     x1, 1, 1
ld          0
setpos(d)   x1, srcdiff
ld          pos_drag
setmode     x1, drive
```

Variable is reference source:

```
ld          srcvar
setsrctyp   x1
ld          variable
setsrcvar   x1
ld          source
setnorm     x1, 1, 1
ld          0
setpos(d)   x1, srcdiff
ld          pos_drag
setmode     x1, drive
```

3.5.2 Rotation monitoring

The following controller library functions are relevant for programming the rotation monitoring feature:

Function	**Meaning**
setmode	Activate rotation monitoring and set encoder input
setdrive	Set encoder input for rotation monitoring
setpos(d)	Input positions for rotation monitoring
getpos(d)	Retrieve positions for rotation monitoring
getsig_sr	Retrieve signal for contouring error

To avoid position errors, rotation monitoring can be activated. In this process, the actual position of the motor is detected by an encoder and compared with the setpoint. If the difference (following error) exceeds the following error limit set by “setpos” or “setposd”,the motor is decelerated. In the controller, a signal flag (“dragerr”) is set for the contouring error; the signal flag can be retrieved with “getsig_sr”. The motor cannot move until the encoder position is within the following error limit range using “setpos” or “setposd” and the signal flag for the contouring error is reset.

The following error limit depends on the encoder resolution and the evaluation of the encoder signal programmed with “setmode”. The following values should be programmed for the following error limit using the “setpos” function:

Encoder resolution	**Encoder evaluation**		
	4 x	**2 x**	**1 x**
1000 marks	72	36	18
500 marks	36	18	9

Example:
For an encoder with a resolution of 1000 marks and single evaluation (equivalent to 1000 increments per revolution), the following error limit 18 should be set.

Programming example for rotation monitoring via encoder connection 1 (e.g. for WDP5-318)

```
ld          watch
getensig    x1
or          dragerr
ensig       x1, watch
ld          encsingle
setmode     p1, encoder
ld          indexer
setnorm     p1, 1, 1
ld          x1
setdrive    p1, 1
ld          18
setpos      p1, maxveldrag
ld          watchdrive
setmode     p1, drive
ld          dragerr
or          motortemp
clrsig_sr   x1, watch
```

Programming example for rotation monitoring on external power controller (e.g. for WP-311)

```
ld          watch
getensig    x1
or          dragerr
ensig       x1, watch
```

3.5.3 Index pulse evaluation

The index pulse (encoder signal I or C) is a signal which is generated once per motor revolution by the encoder. The index pulse can be evaluated as follows:

– As an index signal using the “getsig”, “getsig_sr” and “clrsig_sr” functions (only possible with encoder input 2).
The signal can be used to determine when the motor has turned by one revolution.
– As a trigger signal (see “setmode” function).
The signal can be used for storing the current position after each motor revolution; this position can then be interrogated using the “getpos” or “getposd” function.
– As a stop signal (see “setmode” function).
The signal can be used for aligning the drive with the index pulse, e.g. after a reference movement.

NOTE
Refer to chapter “Store actual position”.

3.5.4 Position control with WDP3-337 and WDP3-338

With the WDP3-337 and WDP3-338 units, the encoder connection 1 (resolver connection) is used for detecting the shaft's actual position and passing it to the internal position controller (see figure 3-14). For this purpose, the encoder connection 1 is automatically set to quadruple evaluation. For a resolver with a resolution of 1024 marks, this results in an internal resolution of 4096 encoder units (increments) per revolution. The default setting of the contouring error limit is 4096 encoder units. This can be changed with the "setpos" or "setposd" functions. If the following error (the difference between setpoint and actual position) exceeds the contouring error limit, the motor is deenergized.

ATTENTION
When the motor is deenergized, it does not have any holding torque. This may result in damage to mechanical components. When working with suspended loads, a brake must be installed on the drive.

A PID position controller is used in the WDP3-337 and WDP3-338. The control parameters can be input with the "setparam" function and interrogated with the "getparam" function. For more information on how to set the control parameters, see the controller manual.
The controller is active when the axis is not "blocked"; see "getstate" function. If the axis is "blocked", it must be released using the "clrsig_sr" function.

NOTE
The setup software ONLINE3 is a convenient tool for adjusting the controller.
The parameter values determined with ONLINE3 should be passed to the position controller using "setparam" in the application program.

The encoder connection 2 can be used for reference variable input in position following mode; see chapter 3.5.1.

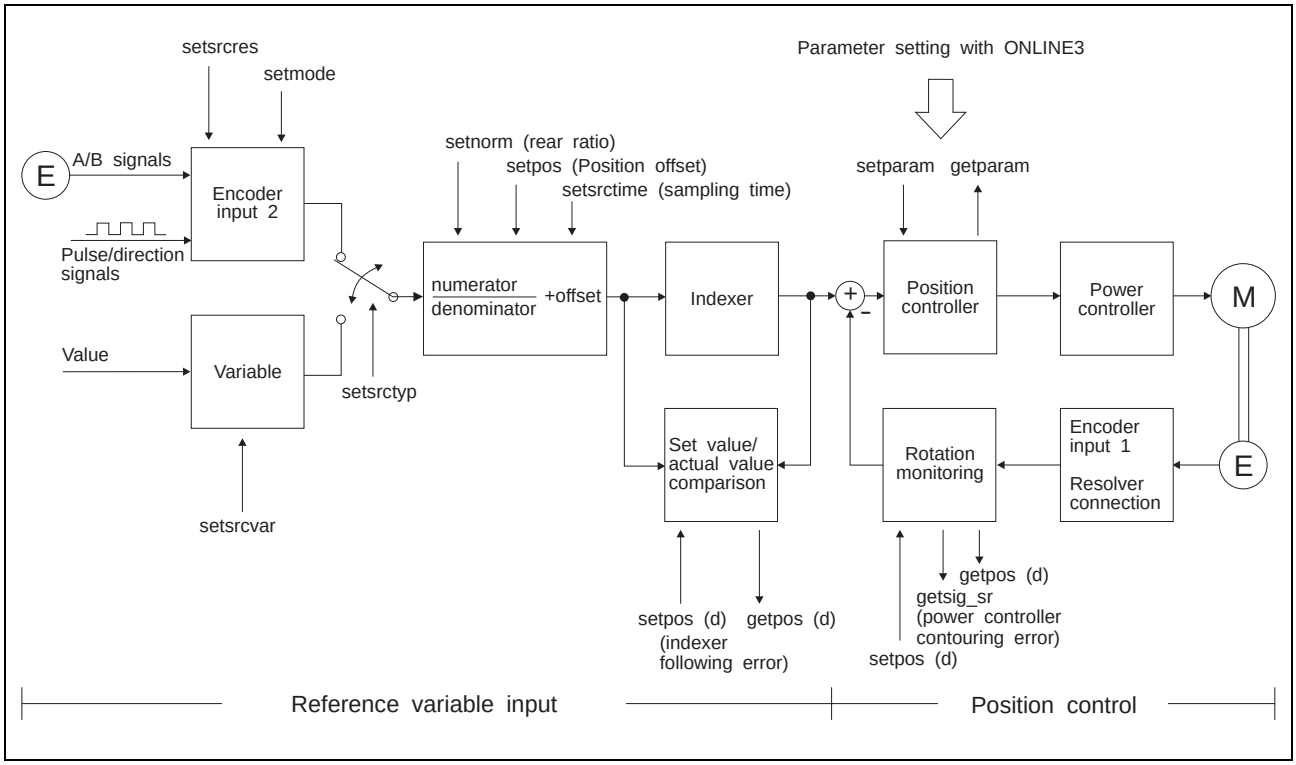


Fig. 3-14 Position control

3.6 Interpolation with multi-axis positioning units

With Series 300 multi-axis positioning units (e.g. WPM-311), several axes can be controlled simultaneously. Axis movements can be performed independently or interdependently (interpolated). At present, linear interpolation is possible with two or three axes. In future releases, additional interpolation methods will be implemented.

NOTE
Linear interpolation can be effected with absolute values (position values referring to the zero points of the axes) or with relative values (position values referring to the current axis position), with or without a waiting period (program execution is suspended until interpolation has been completed).

The following interpolation functions are included in the controller library:

Function	**Meaning**
linpos2(w)	Absolute linear interpolation with two axes (with or without waiting)
linpos3(w)	Absolute linear interpolation with three axes (with or without waiting)
linmove2(w)	Relative linear interpolation with two axes (with or without waiting)
linmove3(w)	Relative linear interpolation with three axes (with or without waiting)

The “stop” function can be used for stopping a linear interpolation process.
In addition, the “geterror_sr” and “getstate” functions can be used for retrieving error information and status conditions when performing linear interpolation.

The following points must be observed for interpolation:

– The axes involved in interpolation must be set to point-to-point mode (see “setmode” function in controller library).

– The speeds and accelerations of the axes involved must be set before the interpolation process (see “vel” and “accel” functions in controller library).

– The target positions, speeds and accelerations of the axes involved cannot be changed during an interpolation process.

– You cannot perform several interpolation processes at the same time.

– An electronic gear is affected by interpolation (it may be necessary to reinitialize it after the interpolation process).

Principle of linear interpolation

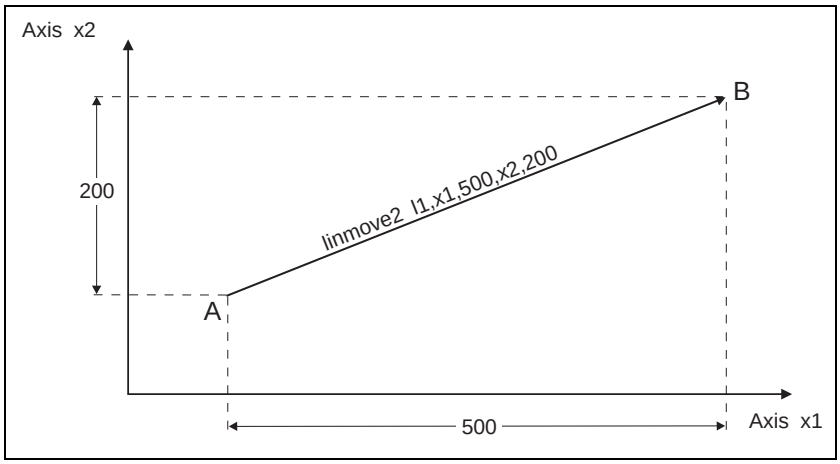


Fig. 3-15 Principle of linear interpolation

Figure 3-15 illustrates the principle of linear interpolation by way of an example:

Two axes (x1 and x2) are to move from point A to point B with linear interpolation.
When the interpolation function is called (e.g. linmove2 l1, x1, 500, x2, 200), the setpoints of the axes are passed and the interpolation is initiated. The linear interpolator (l1) uses the setpoints to calculate the speeds and accelerations required for the interpolation and controls the individual axes. It is ensured that the preset speeds and accelerations of the individual axes are not exceeded.

3.7 Power controller initialization

The first step in movement programming is to switch on the power controller.

Example:

```
Network 1       Switch on power controller
ld              ampon
sethardware     x1

Network 2       Wait for power controller ready status
loop:
ld              watch
getsig          x1
and             ampready
jmpn            loop
```

3.7.1 Activating after power-on

The examples below describe activation of the power controller after having switched on the 24 V supply voltage.

NOTE
Rotation monitoring cannot be set up before having activated the power controller.

NOTE
If a monitoring function is to determine whether the power controller is active, e.g. for controlling a brake, the following programming examples can be used.

Example for WDP-318 and WP-311:

```
ld              watch
getsig_sr       x1
and             ampnotready
jmpn            LT_is_active
```

Example for WDP-337 and WDP-338:

```
ld              watch
getsig_sr       x1
and             ampnotready
eq              0
and(            ampon
gethardware     x1
)
jmpc            LT_is_active
```

On WDP-337 and WDP-338 units, “ampnotready” indicates power controller failure but it does not indicate power controller active status.

NOTE
A brake can also be controlled using the “brake” function.

3.7.2 Activating after power failure

The following sections describe how to activate the power controller after a power failure.

For WDP-318 and WP-311 units without motor position detection via encoder connection p2

See chapter 3.7.1.

For WDP-318 and WP-311 units with motor position detection via encoder connection p2

This connection continues to operate after a power controller power failure using the 24 V supply voltage. This means that the motor position is retained in case of power failure.

Example:

```
ld              ampon
sethardware     x1

;"loop": Wait until power controller is active.
loop:
ld              watch
getsig          x1
and             ampready
jmpn            loop


ld              actual
getpos          p2

;Store position in auxiliary variable
st              actual_value

setpos          x1,actual
ld              actual_value
setpos          p2,actual
ld              ampnotready
or              dragerr
clrsig_sr       x1
ld              err_all
clrerror_sr     x1
```

ATTENTION
It is indispensable to adhere to this order for the following reasons: Setting the position value x1 causes the distance of position values x1 and encoder p2 to be retained. The position values x1 and p2 do not match until the encoder position p2 is set.

NOTE
On WDP-337 and WDP-338 units, "clrsig_sr" may only be used after positional correction since otherwise the position controller would immediately compensate for the existing deviation.

3.8 Controlling a brake

With the “brake” function, any output Qx can be used for controlling a brake. Figure 3-16 shows the relationship between the ENABLE (power controller enable) and READY (power controller ready) signals and the output signal for the brake.

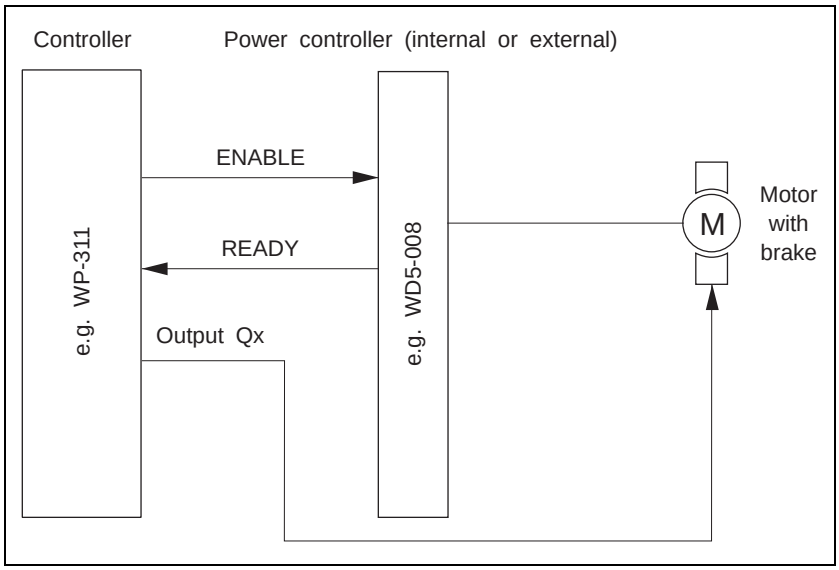


Fig. 3-16 Brake function signals

The brake (fig. 3-17) opens (Qx = high) when the power controller has been enabled and the power controller is ready. The brake is applied (Qx = low) when the power controller is no longer ready (READY = low).

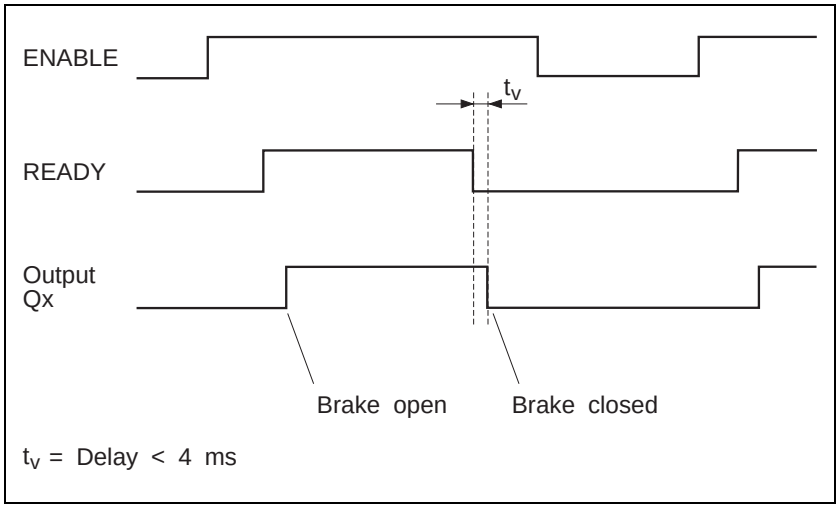


Fig. 3-17 Brake function timing diagram

Function	**Meaning**
brake	Define output for brake

4 Interface programming

The serial interface 1 or 2 on the controller can be used for communication with other devices, e.g. the FT 2000 operating terminal. The interface is freely programmable.

4.1 Serial interface presettings

Before any data can be transmitted, the interface must be initialized and enabled. This is accomplished using the following controller library functions:

Function	**Meaning**
com_init_asc	Initialize serial interface
sethardware	Enable/disable interface

The “com_init_asc” function can be used to set:

– Size of transmit and receive buffer
– Baud rate and data format
– Software handshake (XON/XOFF protocol)
– Break detection

The “sethardware” function is provided for switching the RS 485 output drivers ON or OFF (high resistance). This makes it possible to integrate the controller into a network.

NOTE
After controller power-on, the RS 485 output drivers are always OFF. For data output, however, the drivers must be ON.

NOTE
The “gethardware”, “geterror_sr” and “getstate” functions can be used to retrieve various status conditions of the interface.

4.2 Data transmission through the serial interface

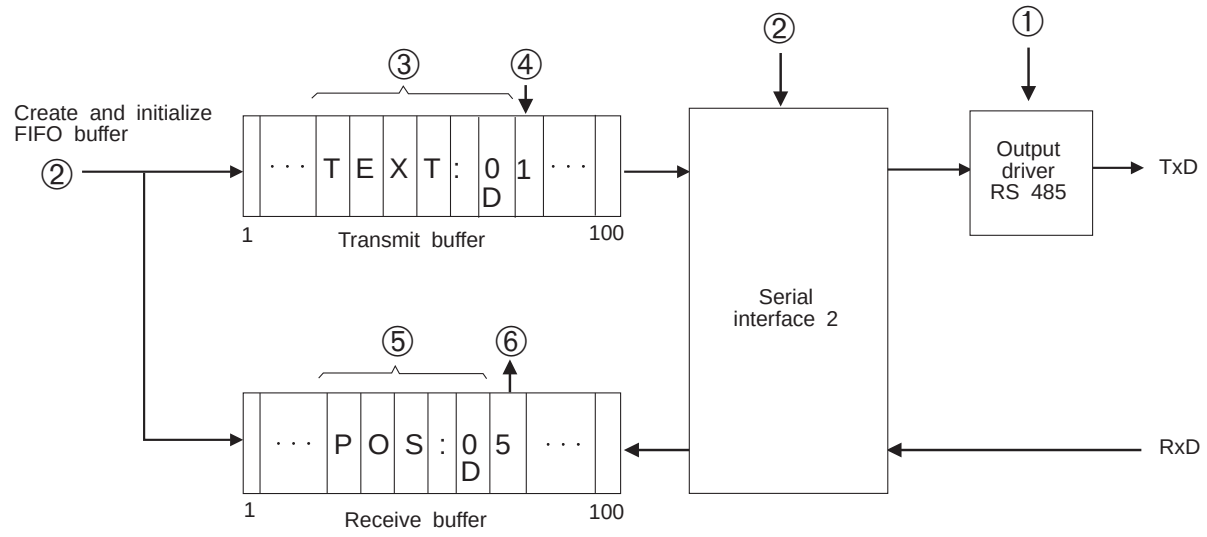
The serial interface can be used to transmit individual characters or strings with the following functions:

Function	**Meaning**
send_char	Transmit single character
send_string	Transmit string
receive_char	Receive single character
receive_string	Receive string

The characters are transmitted as follows:

Characters in I/O buffer	**Sense of transmission**	**Characters through interface,**	
ASCII characters (16#20 - 16#7E)	↔	ASCII characters (16#20 - 16#7E)	
Other characters (16#7F - 16#FF)	↔	16#7F - 16#FF	
$$	↔	$ (16#24)	*
$’	↔	’ (16#27)	*
$L	↔	<LF> (16#0A) Linefeed	*
$N	↔	(16#0A) Newline	*
$P	↔	<FF> (16#0C) Formfeed	*
$R	↔	<CR> (16#0D) Carriage-Return	*
$T	↔	<TAB> (16#09) Tabulator	*
$00 - $1F	←	16#00 - 16#1F	*
$00 - $FF	→	16#00 - 16#FF	*

* Transmission of these characters involves character conversion.




```
VAR
receive_char    BYTE            0
send_buf        STRING  (7)     'TEXT:$R'
receive_buf     STRING  (100)   ' '
com_status      WORD            0
VAR_END
```

Block header

```
① ld            txd_on
  sethardware   c2
```

Enable interface

```
② ld            100
  com_init_asc  c2, 9600, 7, 1, even, no, com_status
  :
```

Initialize interface

```
③ ld            0
  send_string   c2, send_buf, com_status
  :
```

Transmit ASCII string "TEXT:<CR>"

```
④ ld            16#31
  send_char     c2, com_status
  :
```

Transmit ASCII character "1"

```
⑤ ld            16#0D
  receive_string c2,receive_buf, com_status
  :
```

Receive ASCII string

```
⑥ ld            0
  receive_char  c2,receive_char, com_status
```

Receive ASCII character

Fig. 4-1 Interface programming example

4.3 String functions

String functions are used for formatting and interpreting character strings. They can be transmitted, for example, through the serial interface 2.

4.3.1 String formatting functions

The following string formatting functions are included in the controller library:

Function	Meaning
sf_string	Insert text into string
sf_dint	Convert integer to ASCII number and insert into string
sf_lreal	Convert real number to ASCII number and insert into string

4.3.2 String interpreter functions

The following string interpreter functions are included in the controller library:

Function	Meaning
si_comp	Compare text with string
si_dint	Find ASCII number in a string and convert it to integer
si_lreal	Find ASCII number in a string and convert it to a real number
si_string	Find character in a string

The string functions have the following features in common:

- The first input value (value in CR) of a string function is always the string position where the function begins to work.
- When a string function is executed successfully, it always returns the string position where it stopped its activity. This position can be used as the start position for the next string operation.
- If an error occurs during execution of a string function, it returns the code “-1” (value in CR).
- If the first input value (value in CR) of a string function is ≤ 0, the value is decremented (e.g. CR = -1 yields CR = -2). This can be used to program a sequence of string functions with a subsequent error check (e.g. jmpc error_string). The absolute value of the negative number can then be used to determine the function which caused the error.

NOTE
For a description of the individual functions, refer to the block library chapter.

```
VAR
    string1     STRING  (80)      ' '
    text1       STRING            'Position:'
    length      INT               0
    pos         DINT              0
    status      WORD              0
    form_std    WORD              0
VAR_END


Network 1       Read current position
ld              actual
getpos          x1
st              pos

Network 2       Format string
ld              1
sf_string       string1, text1
sf_dint         string1, pos, length, form_std
le              0
jmpc            stringformat_error

Network 3       Send string
ld              0
send_string     c2, string1, status
jmpn            interface_error
```

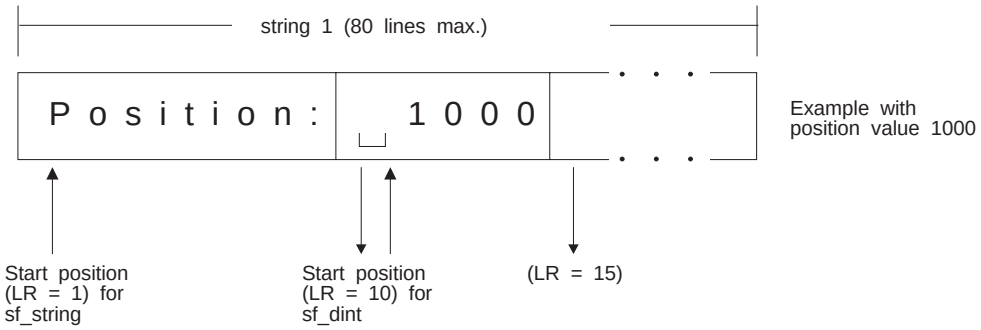


Fig. 4-2 String formatting programing example

5 Error messages and troubleshooting

While developing a program with the programming system the following error types may occur:

- DOS operating system errors
- Programming errors (syntax errors)
- Errors during data transmission to controller
- Program execution errors (runtime errors)

Any errors detected are displayed in a message on the PC screen; see BPRO3 Operating Manual.

Controller error memory

Error memory

System errors

Runtime errors are recorded in the controller error memory and displayed on the status display of the controller. A maximum of 16 errors can be stored in the controller error memory (the first 8 and the last 8 errors which occurred). With the programming system, the contents of the error memory and a detailed description of the errors can be displayed; see BPRO3 Operating Manual. The errors recorded in the error memory, except system errors, are cleared when “Reset controller” is invoked or the application program restarted.

Error classes

Error class characteristics

Runtime errors are structured according to error classes. The various error classes are characterized by the error type and the effect on the controller.

Error class/ Significance	**Controller response**	**Rectification**
Error class 0 System error	STOP status, RUN status not available. The error is stored in the error memory and can only be cleared by booting.	Call BERGER LAHR
Error class 1 Fatal error in application program	STOP status, RUN status initially available. The error is stored in the error memory.	Modify and reload application program
Error class 2 Non-fatal error in application program	STOP status, RUN status available. The error is stored in the error memory.	See troubleshooting table in BPRO3 Operating Manual.
Error class 3 Setting error	STOP status, RUN status available. The error is stored in the error memory.	See troubleshooting table in BPRO3 Operating Manual.
Error class 4 Programming error	The application program continues to execute. The error is stored in the error memory and registered in the resource error word. The resource error word can be read from the application program with the “geterror_sr” function.	See troubleshooting table in BPRO3 Operating Manual.
Error class 5 Signal monitoring	The application program continues to execute, the drive movement is stopped. Any active signal is registered in the resource signal word and can be read from the application program with the “getsig_sr” function. The error is stored in the error memory.	Can be defined by the user.

Programmable error handling and signal monitoring

Programming errors (bugs)

The “geterror_sr” function of the controller library can be used to handle programming errors (bugs) in the application program.

Internal signal states

In addition, the “getsig_sr” function can be used for retrieving internal signal states required for controller operation.

```
Network 10                  Error handling
ld                          err_all
geterror_sr                 x1
andn                        err_watch, err_w_extern
jmpn                        end_eval
ld                          drive
stop                        x1


Network 11                  Signal monitoring
end_eval:
ld                          watch
getsig_sr                   x1
andn                        limp
andn                        limn
jmpc                        Limit switch handling
```

Fig. 5-1 Error handling and signal monitoring example

6 Project development

This chapter provides a simple step-by-step example for the procedure involved in developing a project.

Task to be completed

A motor is to perform a movement, alternating to the right and left. The end of the positioning operation is to be reported via an output.

Programming with BPRO3

The IL program for this task may be as follows:

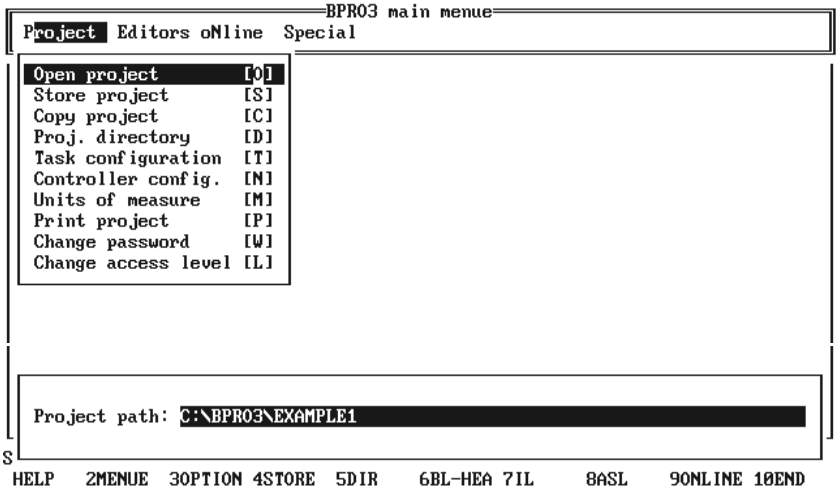
```
Network 1    Move at 2000 Hz, 1000 steps
loop:
ld           2000
vel          x1

ld           1000
movef        x1

Network 2    Switch on output 2
ld           1
st           output2

Network 3    Move at 5000 Hz, -1000 steps
ld           5000
vel          x1

ld           -1000
movef        x1

Network 4    Switch off output 2, wait for 1 s
             and jump to  "loop"
ld           0
st           output2

ld           T#1s
wait_time

jmp          loop
```

This program is to be assigned to the SEQUENCE task, and the axis is to be initialized in the INIT task. The PLC task is not required.

1. Opening a project

Enter the desired project name. If the project does not yet exist, a new project is generated. In this process, the empty program blocks INIT_PRG, SEQUENCE_PRG and PCL_PRG are created and assigned to their respective tasks (INIT, SEQUENCE, PLC task). These program blocks can be programmed when required.

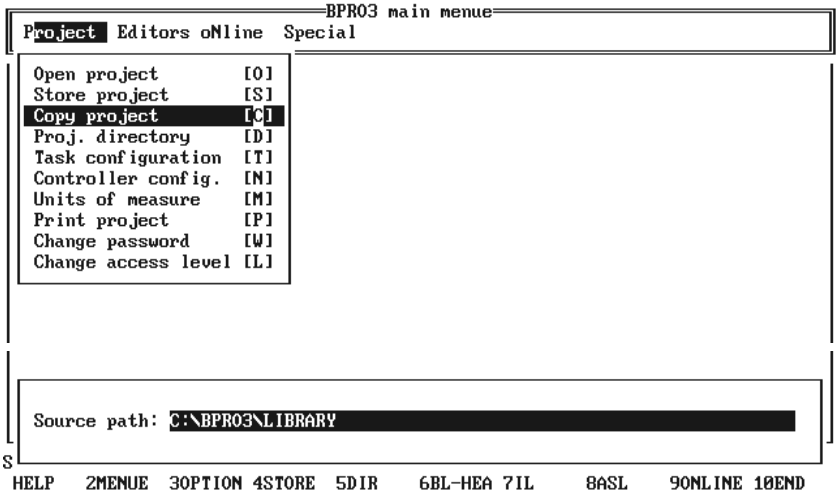
NOTE
"Open project" may take several minutes, depending on the type of computer used.



2. Copying the "LIBRARY" project (user library) into the open project

The user library contains useful blocks for programming (see thumb index section 3 for a description).

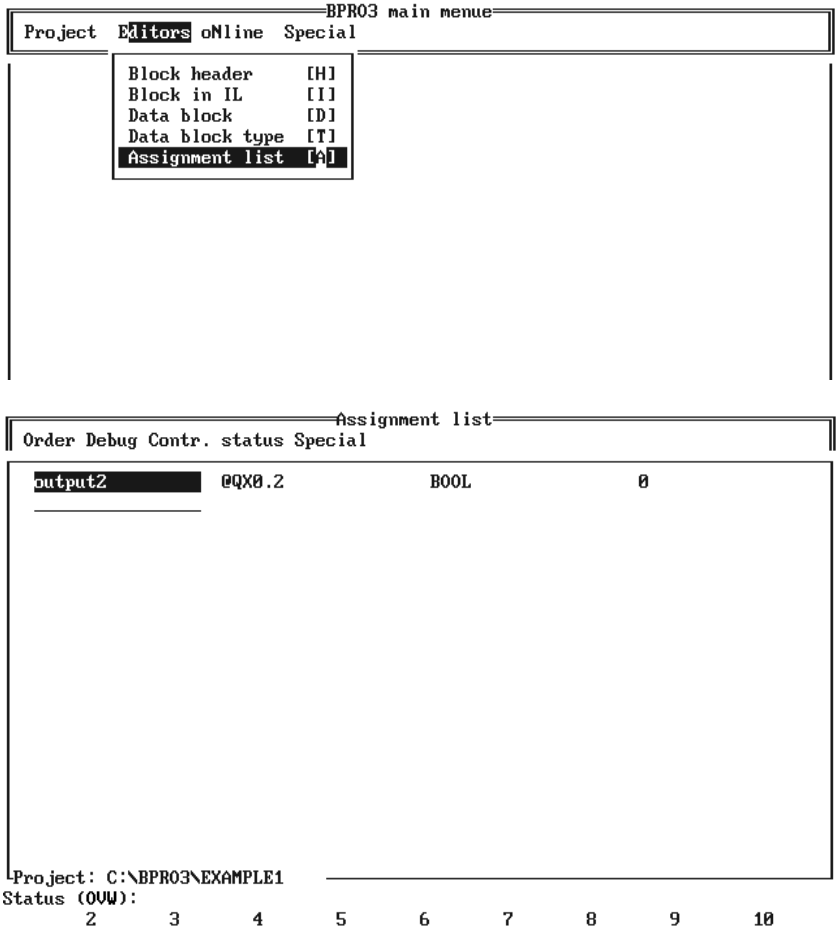
ATTENTION
The opened project is overwritten when "Copy project" is selected.



3. Creating an assignment list

The inputs/outputs or flags may be assigned optional symbolic names in the assignment list.

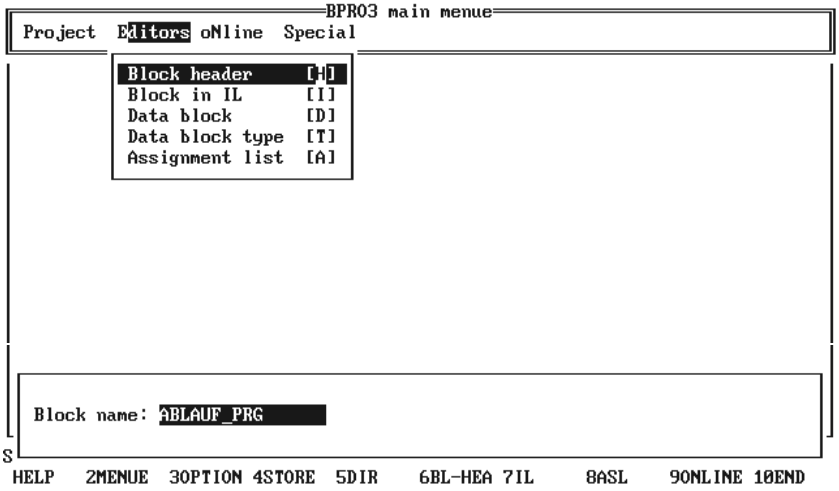
NOTE
When selecting "Open project", all controller system constants are automatically entered into the assignment list (see chapter 8.1).

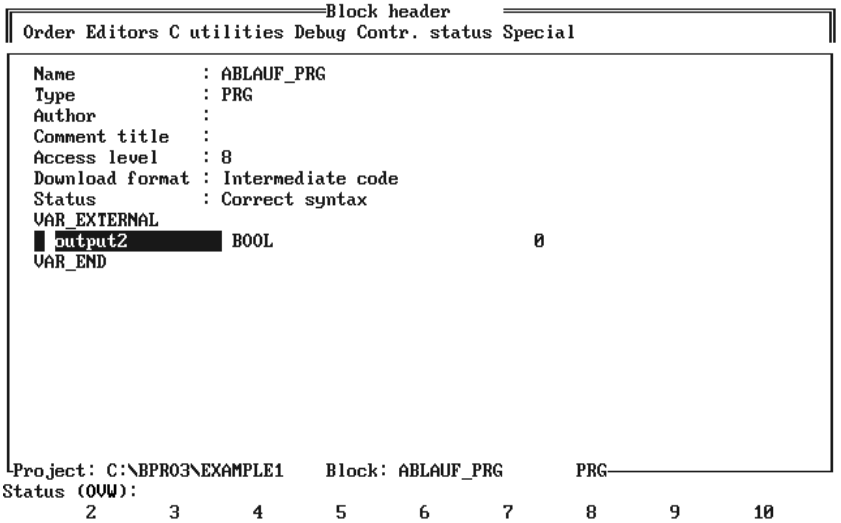


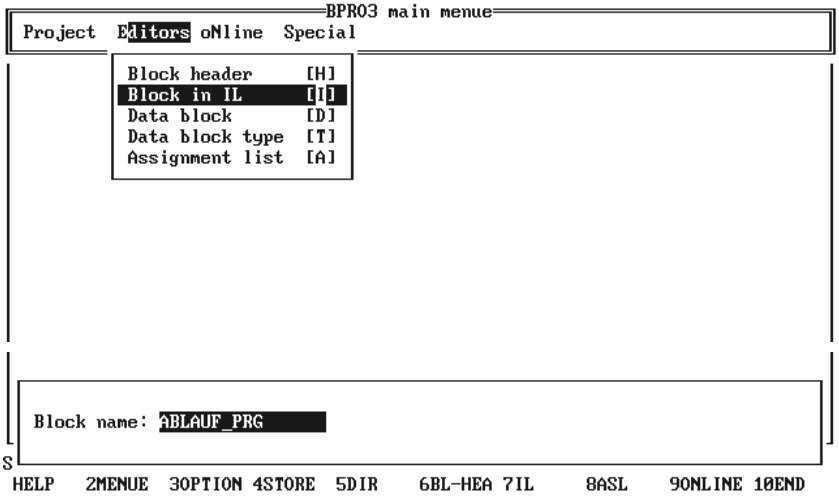
4. Programming blocks

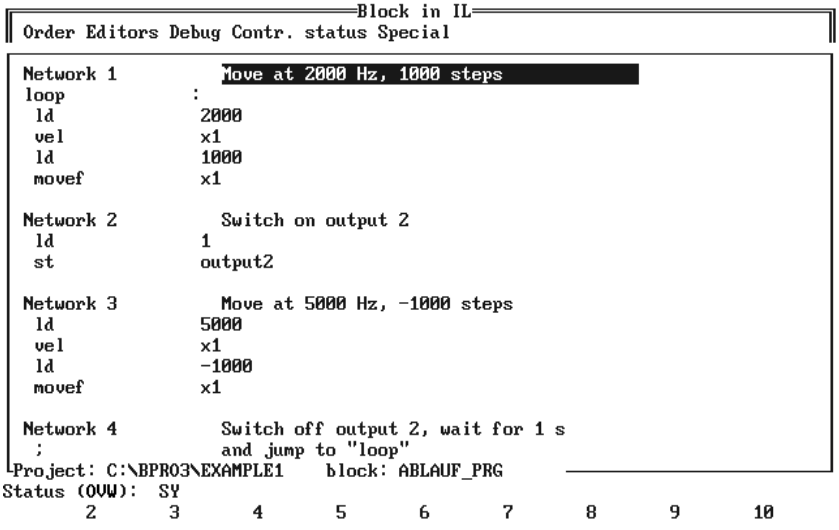
This step involves programming the program blocks INIT_PRG, SEQUENCE_PRG, PLC_PRG and possibly other blocks.

NOTE
If symbolic names are used for inputs/outputs or flags (see step "3. Creating an assignment list"), they must be entered in the block header as VAR_EXTERNAL.

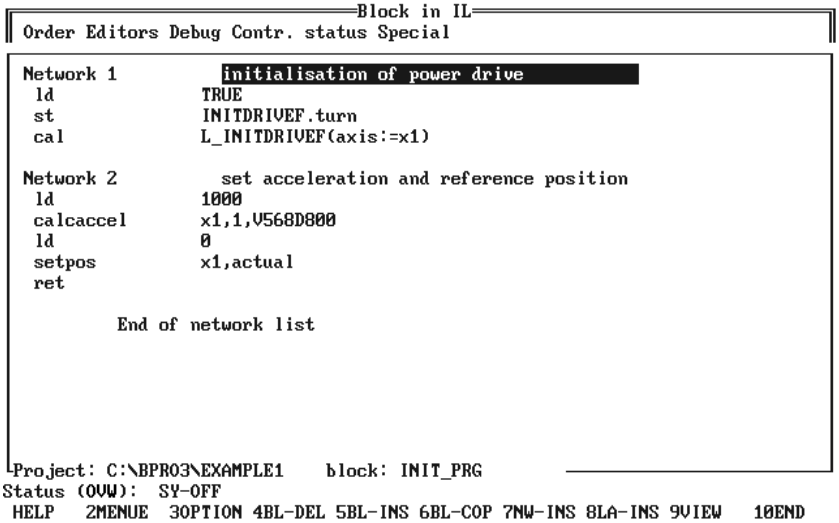








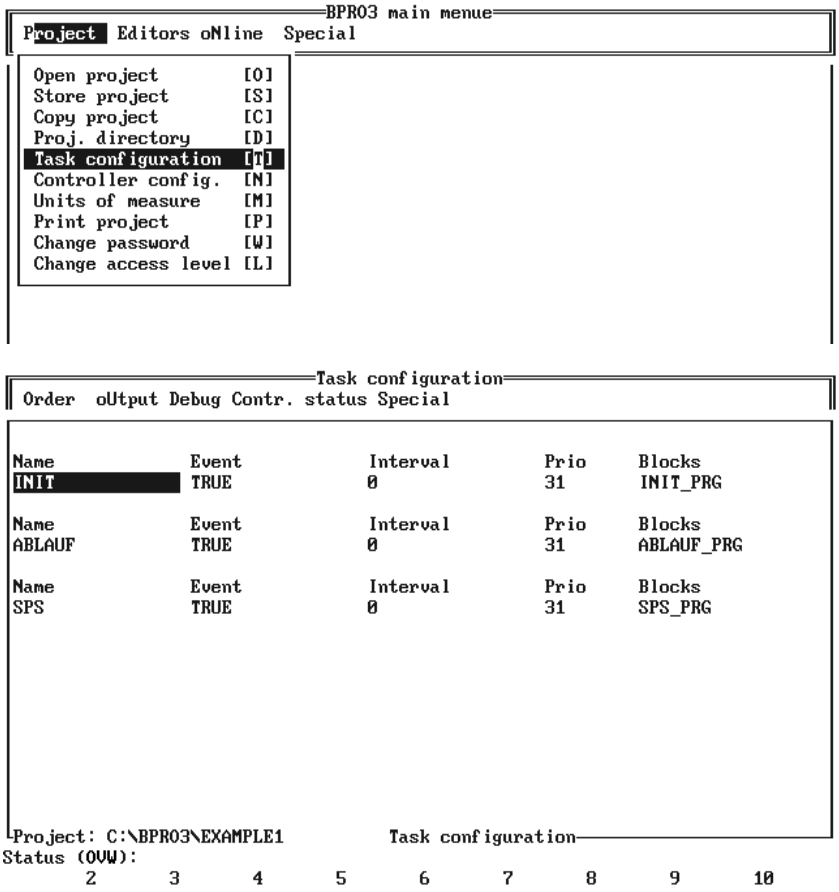
The axis is initialized in the program block INIT_PRG.



5. Assigning program blocks to tasks

With the "Task configuration" option, you can assign one or more program blocks to the individual tasks.

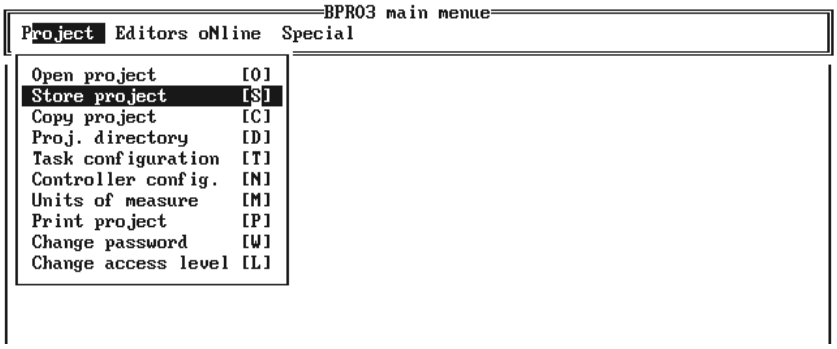
NOTE
When opening a new project, the empty program blocks INIT_PRG, SEQUENCE_PRG and PLC_PRG are automatically created and assigned to their respective tasks.



6. Storing a project

When selecting the “Store project” option, BPRO3 overwrites the original project data with the backup copies which are created automatically.

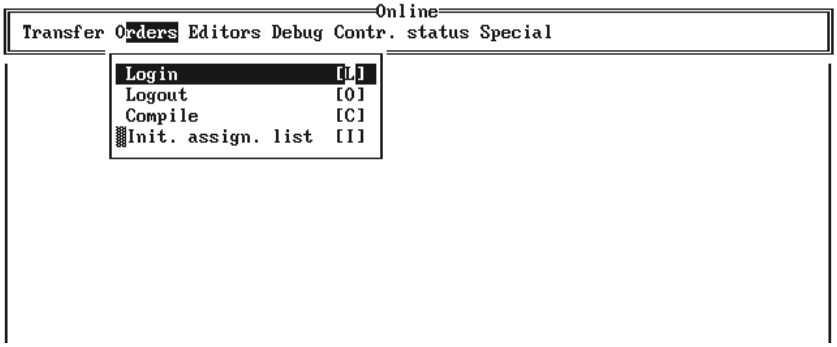
NOTE
After a power failure the inputs of the last session can be repeated automatically; see BPRO3 Operating Manual, chapter 3.2.4, "Log files".



7. Establishing the connection to the controller

Establish the connection to the controller by selecting the “Login” option if the controller is correctly wired to the PC and has been switched on.

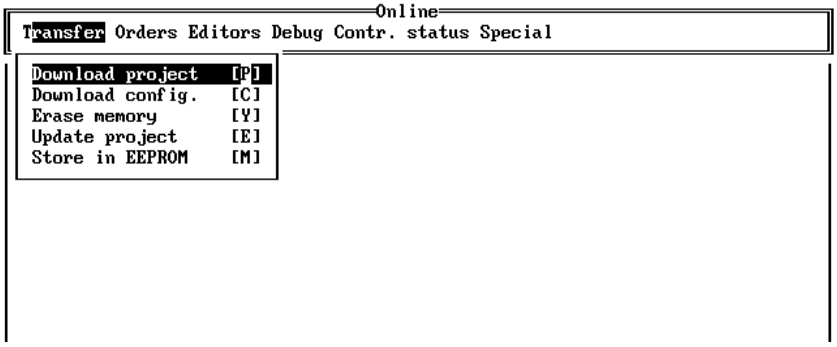
NOTE
The default interface on the PC is COM1. You can change this by selecting the “Special/Setup” option from the BPRO3 main menue.



8. Loading a project into the controller

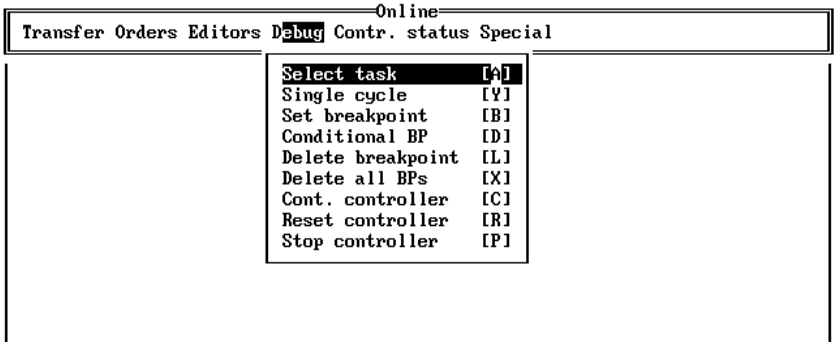
Select “Download project” to load the project into the controller when the controller is in application mode and has RESET status (see controller manual).

NOTE
In order to carry out a debugging run, the blocks of the project must be loaded in pseudo-code (see block header).

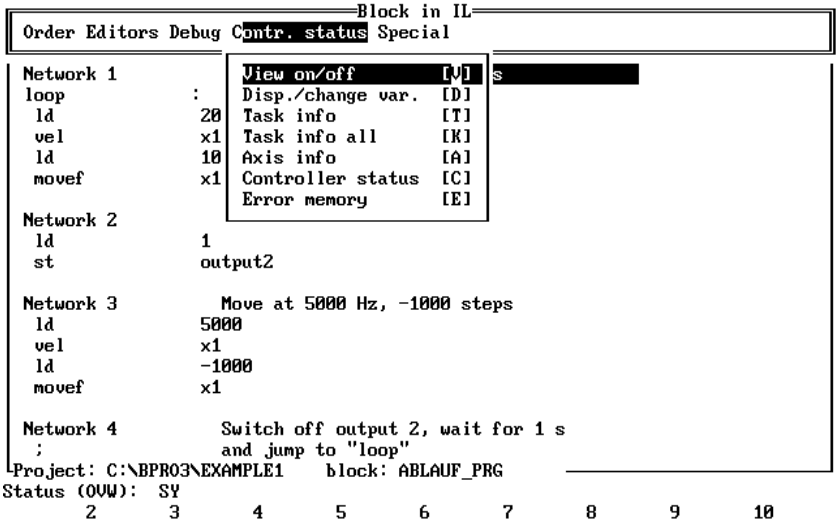


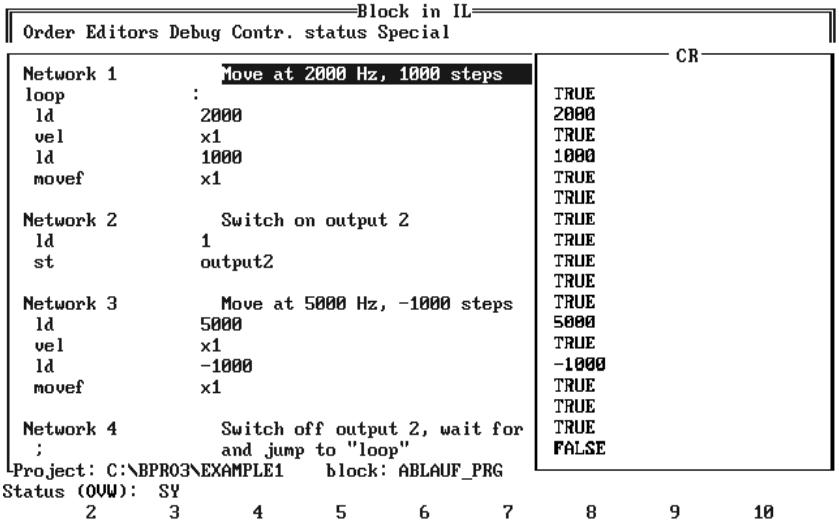
9. Debugging

The pull-down menue “Online/Debug” provides some options for program testing, or debugging.



In addition, you can view the program sequence in the IL editor. The values of the CR (current result) are displayed on the right next to the program instructions.





10. Program run

After debugging, the blocks of the project should be loaded into the controller in object code. This will considerably reduce the time required for program execution.

Start the program by pressing the RUN key on the controller.

7 Block library

The block library is composed of the standard library and the controller library.
Ready-made blocks are available in the block library of the programming system. The blocks to be selected depend on the controller, or the controller configuration. The block library includes:

- Standard library with standard functions and standard function blocks according to the IEC 1131-3 standard
- Controller library (controller-dependent) with controller-specific blocks, e.g. movement functions

The standard library includes the following standard functions (SF_FUN) and standard function blocks (SF_FBS).

Name	SF_FUN	SF_FBS	Meaning	See page
abs_lreal	x		Absolute value	7-7
acos	x		Arc cosine function	7-7
asin	x		Arc sine function	7-8
atan	x		Arc tangent function	7-8
bcd_to_dint	x		Conversion from BCD code to DINT	7-9
cos	x		Cosine function	7-9
ctd		x	Decremental counter	7-20
ctu		x	Incremental counter	7-21
ctud		x	Incremental/decremental counter	7-22
dint_to_lreal	x		Type conversion from DINT to LREAL	7-10
dint_to_time	x		Type conversion from DINT to TIME	7-10
dint_to_word	x		Type conversion from DINT to WORD	7-11
exp	x		Exponential function	7-11
f_trig		x	Trigger for trailing edges	7-23
ln	x		Natural logarithm	7-12
log	x		Common logarithm	7-12
lreal_to_dint	x		Type conversion from LREAL to DINT	7-13
rol	x		Rotate CR to the left	7-13
ror	x		Rotate CR to the right	7-14
r_trig		x	Trigger for leading edges	7-24
rs		x	Reset/set flipflop	7-25
sema		x	Semaphore (not yet implemented)	7-26
shl	x		Shift CR to the left	7-14
shr	x		Shift CR to the right	7-15
sin	x		Sine function	7-15
sqrt	x		Square root	7-16
sr		x	Set/reset flipflop	7-27
tan	x		Tangent function	7-16
time_to_dint	x		Type conversion from TIME to DINT	7-17
time_to_lreal	x		Type conversion from TIME to LREAL	7-17
tof		x	Timer OFF delay	7-28
ton		x	Timer ON delay	7-29
tp		x	Pulse timer	7-30
trunc	x		Round off numbers of type LREAL	7-18
word_to_dint	x		Type conversion from WORD to DINT	7-18

In the controller library, the following control functions are available for programming the controller.

Controller function	Meaning	See page
accel	Select acceleration/deceleration curve	7-35
acclin	Calculate and activate linear acceleration/deceleration curve	7-36
brake	Define output for brake	7-37
calcaccel	Calculate acceleration/deceleration curve	7-38
clrerror_sr	Clear error condition	7-39
clrsig_sr	Clear temporarily stored axis signals, enable axis	7-40
	Clear temporarily stored encoder signals	7-41
com_init_asc	Initialize a serial interface for ASCII mode	7-42
continue	Continue interrupted axis movement	7-43
cycletime	Set PLC cycle time monitoring	7-44
debug	Reset outputs and stop axes at controller stop ON/OFF	7-45
display	Display number on controller's LED display	7-46
ensig	Enable or disable axis signals	7-47
getaccel	Get ramp assignment	7-48
getaccfac	Get maximum acceleration of a ramp	7-49
getanalog	Get analog value of an input channel from an analog module	7-50
getcurrent	Get electrical current values	7-51
getensig	Get enabled axis signals	7-52
geterror_sr	Get axis error condition	7-53
	Get encoder error condition	7-54
	Get linear interpolator error condition [2)]	2-55
	Get serial interface error condition	7-56
	Get mathematical error condition	7-57
gethardware	Get hardware signals	7-58
getmode	Get axis or encoder operating mode	7-59
getnorm	Get normalizing factors (axis)	7-60
	Get normalizing factors (encoder)	7-61
getparam [1)]	Get control parameters	7-62
getpos	Get axis positions in user-defined units	7-63
	Get encoder positions in user-defined units	7-64
getposd	Get axis positions in drive units	7-65
	Get encoder positions in encoder units	7-66
getsig	Get current axis signals	7-67
	Get current encoder signals	7-68
getsig_activ_h	Get active state of axis signals	7-69
getsig_sr	Get temporarily stored axis signals	7-70
	Get temporarily stored encoder signals	7-71

Controller function	Meaning	See page
getstate	Get axis status	7-72
	Get linear interpolation status conditions [2)]	7-75
	Get serial interface status	7-77
	Get battery status	7-78
	Get front panel key status	7-79
getvel	Get speed values	7-80
linmove2 [2)]	Relative linear interpolation with two axes	7-81
linmove2w [2)]	Relative linear interpolation with two axes and wait for end of movement	7-82
linmove3 [2)]	Relative linear interpolation with three axes	7-83
linmove3w [2)]	Relative linear interpolation with three axes and wait for end of movement	7-84
linpos2 [2)]	Absolute linear interpolation with two axes	7-85
linpos2w [2)]	Absolute linear interpolation with two axes and wait for end of movement	7-86
linpos3 [2)]	Absolute linear interpolation with three axes	7-87
linpos3w [2)]	Absolute linear interpolation with three axes and wait for end of movement	7-88
list	Disable position list processing	7-89
move	Relative positioning in user-defined units	7-90
moved	Relative positioning in drive units	7-91
movedf	Relative positioning in drive units and wait until position reached	7-92
movedw	Relative positioning in drive units and wait for end of movement	7-93
movef	Relative positioning in user-defined units and wait until position reached	7-94
movew	Relative positioning in user-defined units and wait for end of movement	7-95
pos	Absolute positioning in user-defined units	7-96
posd	Absolute positioning in drive units	7-97
posdf	Absolute positioning in drive units and wait until position reached	7-98
posdw	Absolute positioning in drive units and wait for end of movement	7-99
posf	Absolute positioning in user-defined units and wait until position reached	7-100
posw	Absolute positioning in user-defined units and wait for end of movement	7-101
receive_char	Read single character from transmit and receive buffer	7-102
receive_string	Read string from transmit and receive buffer	7-103
record	Start recording	7-104
refpos	Reference movement	7-105
refposf	Reference movement and wait until reference point reached	7-106
refposw	Reference movement and wait for end of movement	7-107
send_char	Output single characters to serial interface	7-108
send_string	Output string to serial interface	7-109
setaccsig	Signal-dependent deceleration	7-110
setanalog	Write analog value to an input channel of an analog module	7-111
setcurrent	Set electrical current values for various movement states	7-112

Controller function	Meaning	See page
setdrive	Set encoder input for position feedback	7-113
sethardware	Hardware settings for a serial interface	7-114
	Hardware settings for the axes	7-115
setmode	Set axis operating mode	7-116
	Set encoder operating mode	7-117
setnorm	Set normalizing factors for axis	7-118
	Set normalizing factors for encoder	7-119
setnormlist	Enable position list for gear ratios	7-120
setparam [1)]	Set control parameters	7-121
setpos	Set axis positions in user-defined units	7-122
	Set encoder positions in user-defined units	7-123
setposd	Set axis positions in drive units	7-124
	Set encoder positions in encoder units	7-125
setsig_activ_h	Set active state of axis signals	7-126
setsiglist	Enable position list for signal output	7-127
setsrcres	Set encoder input for position following mode	7-128
setsrctime	Set sampling time for position following mode	7-129
setsrctyp	Set reference variable type for position following mode	7-130
setsrcvar	Set variable for position following mode	7-131
setvel	Set start and maximum speeds	7-132
setvellist	Enable position list for speeds	7-133
sf_dint	Convert DINT to STRING and insert into string	7-134
sf_lreal	Convert LREAL to STRING and insert into string	7-136
sf_string	Insert or replace text	7-138
si_comp	Compare text	7-139
si_dint	Convert number string to DINT type	7-140
si_lreal	Convert number string to LREAL type	7-141
si_string	Extract substring	7-142
stop	Stop axis movement, stop controller	7-143
vel	Set the set speed	7-144
wait_time	Wait command for SEQUENCE programs	7-145

1) This function is only available on units with position controller: e.g. WDP3-337 and WDP3-338
2) This function is only available on multi-axis systems: e.g. WPM-311

7.1 Standard library

The standard library contains the standard functions and the standard function blocks . The blocks to be selected depend on the controller, or the controller configuration.

The following sections describe the standard functions and standard function blocks of the standard library as follows:

Description: A brief description of the block	
Block header: Parameter declaration	***FBD representation*** Symbol for FBD representation
Programming example: Simple programming example for the block call	***Comment:*** Description of the commands used

Parameter passing is effected when the block is called. The first parameter is made available in the CR. Any other parameters are specified with the block call.

NOTE
When specifying parameters, strictly adhere to the sequence of their declarations in the block to be called.

7.1.1 Standard functions

The standard functions can be used by any block. A function can have several input variables but always returns only one result. The function result is passed to the calling block via the CR. With each function call, the local variables of functions are re-initialized. Identical input for functions will always return the same result.

The standard functions are listed in the table and described in detail in the sections below.

Standard functions	**FUN name**	**Meaning**	**See page**
General functions	abs_lreal	Absolute value of a real number	7-7
	sqrt	Square root	7-16
Logarithmic functions	exp	Exponential function	7-11
	ln	Natural logarithm	7-12
	log	Common logarithm	7-12
Trigonometrical functions	acos	Arc cosine function	7-7
	asin	Arc sine function	7-8
	atan	Arc tangent function	7-8
	cos	Cosine function	7-9
	sin	Sine function	7-15
	tan	Tangent function	7-16
Conversion functions	bcd_to_dint	Conversion from BCD code to DINT	7-9
	dint_to_lreal	Type conversion from DINT to LREAL	7-10
	dint_to_time	Type conversion from DINT to TIME	7-10
	dint_to_word	Type conversion from DINT to WORD	7-11
	lreal_to_dint	Type conversion from LREAL to DINT	7-13
	time_to_dint	Type conversion from TIME to DINT	7-17
	time_to_lreal	Type conversion from TIME to LREAL	7-17
	trunc	Round off numbers of type LREAL	7-18
	word_to_dint	Type conversion from WORD to DINT	7-18
Shifting and rotating functions	rol	Rotate CR to the left	7-13
	ror	Rotate CR to the right	7-14
	shl	Shift CR to the left	7-14
	shr	Shift CR to the right	7-15

The following tables contain the corresponding functions with a description, block header and programming example.

The description is a brief explanation of the function.

In the block header, the name, the type and the input/output variables are defined.

The programming example illustrates IL programming.

The FBD representation shows the functions in a graphic format.

7.1.1.1 abs_lreal, Absolute value

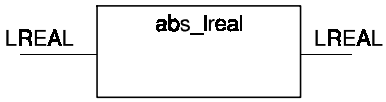
Description:

The “abs_lreal” function supplies the absolute value of a number of type LREAL. The result is of type LREAL.

Block header:

```
Name:          abs_lreal
Type:          STANDARD_FUN

VAR_INPUT
  lreal        LREAL
VAR_END

VAR_OUTPUT
  abs_lreal    LREAL
VAR_END
```

FBD representation



Programming example:

```
ld             - 234.56
abs_lreal
```

Comment:

Load value into CR
Calculate the absolute value (result = 234.56)

7.1.1.2 acos, Arc cosine function

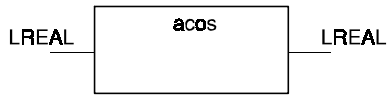
Description:

The “acos” function calculates the arc cosine of the input value. The valid range for the input value is -1.0 ≤ input value ≤ +1.0. After the function call the result is stored as an angle expressed in radians (rad) in the CR. The data type of the result is LREAL. If the input value is not within the valid range, a mathematical error is generated. This error can be determined with the “geterror_sr” function.

Block header:

```
Name:          acos
Type:          STANDARD_FUN

VAR_INPUT
  in           LREAL
VAR_END

VAR_OUTPUT
  acos         LREAL
VAR_END
```

FBD representation



Programming example:

```
ld             - 0.75
acos
```

Comment:

Load value into CR
Function call

7.1.1.3 asin, Arc sine function

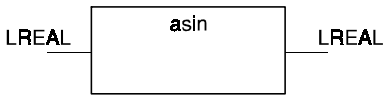
Description:

The “asin” function calculates the arc sine of the input value. The valid range for the input value is -1.0 ≤ input value ≤ +1.0. After the function call the result is stored as an angle expressed in radians (rad) in the CR. The data type of the result is LREAL. If the input value is not within the valid range, a mathematical error is generated. This error can be determined with the “geterror_sr” function.

Block header:

```
Name:          asin
Type:          STANDARD_FUN

VAR_INPUT
  in           LREAL
VAR_END

VAR_OUTPUT
  asin         LREAL
VAR_END
```

FBD representation



Programming example:

```
ld             0.25
asin
```

Comment:

Load value into CR
Function call

7.1.1.4 atan, Arc tangent function

Description:

The “atan” function calculates the arc tangent of the input value. After the function call the result is stored as an angle expressed in radians (rad) in the CR. The data type of the result is LREAL.

Block header:

```
Name:          atan
Type:          STANDARD_FUN

VAR_INPUT
  in           LREAL
VAR_END

VAR_OUTPUT
  atan         LREAL
VAR_END
```

FBD representation

Programming example:

```
ld             20.3
atan
```

Comment:

Load value into CR
Function call

7.1.1.5 bcd_to_dint, Conversion from BCD code to DINT

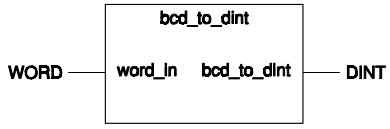
Description:

The "bcd_to_dint" function converts a BCD encoded value of type WORD to DINT. An incorrect input value will return an incorrect output value.

Block header:

```
Name:          bcd_to_dint
Type:          STANDARD_FUN

VAR_INPUT
  word_in      WORD
VAR_END

VAR_OUTPUT
  bcd_to_dint  DINT
VAR_END
```

FBD representation



Programming example:

```
ld          16#3851
bcd_to_dint
```

Comment:

Load BCD code into CR
Function call

7.1.1.6 cos, Cosine function

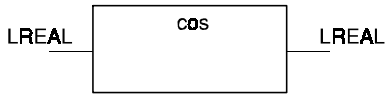
Description:

The “cos” function calculates the cosine of an angle expressed in radians (rad). The valid range for the input value is -65536.0 ≤ input value ≤ +65535.0. The data type of the result is LREAL. If the input value is not within the valid range, a mathematical error is generated. This error can be determined with the “geterror_sr” function.

Block header:

```
Name:          cos
Type:          STANDARD_FUN

VAR_INPUT
  in           LREAL
VAR_END

VAR_OUTPUT
  cos          LREAL
VAR_END
```

FBD representation



Programming example:

```
ld          3.14
cos
```

Comment:

Load radian angle into CR
Function call

7.1.1.7 dint_to_lreal, Type conversion from DINT to LREAL

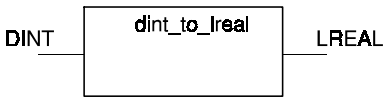
Description:

The “dint_to_lreal” function converts the data type DINT to LREAL.

Block header:

```
Name:           dint_to_lreal
Type:           STANDARD_FUN

VAR_INPUT
  dint_in       DINT
VAR_END

VAR_OUTPUT
  dint_to_lreal LREAL
VAR_END
```

FBD representation



Programming example:

```
ld              123456
dint_to_lreal
```

Comment:

Load value into CR
Function call

7.1.1.8 dint_to_time, Type conversion from DINT to TIME

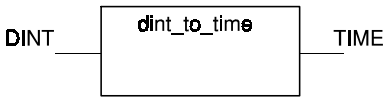
Description:

The “dint_to_time” function converts the data type DINT to TIME.

Block header:

```
Name:           dint_to_time
Type:           STANDARD_FUN

VAR_INPUT
  dint_in       DINT
VAR_END

VAR_OUTPUT
  dint_to_time  TIME
VAR_END
```

FBD representation



Programming example:

```
ld              1234
dint_to_time
```

Comment:

Load value into CR
Function call

7.1.1.9 dint_to_word, Type conversion from DINT to WORD

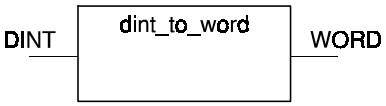
Description:

The “dint_to_word” function converts the data type DINT to WORD.

Block header:

```
Name:           dint_to_word
Type:           STANDARD_FUN

VAR_INPUT
  dint_in       DINT
VAR_END

VAR_OUTPUT
  dint_to_word  WORD
VAR_END
```

FBD representation



Programming example:

```
ld              123456
dint_to_word
```

Comment:

Load value into CR
Function call

7.1.1.10 exp, Exponential function

Description:

The “exp” function calculates the exponential equation

Result = $e^{\text{input value.}}$

If the result is not within the LREAL range of values, a mathematical error is generated. This error can be determined with the “geterror_sr” function.

Block header:

```
Name:           exp
Type:           STANDARD_FUN

VAR_INPUT
  in            LREAL
VAR_END

VAR_OUTPUT
  exp           LREAL
VAR_END
```

FBD representation

Programming example:

```
ld              2.0
exp
```

Comment:

Load value into CR
Function call

7.1.1.11 ln, Natural logarithm

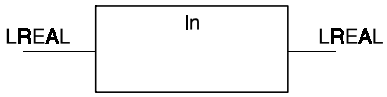
Description:

The “ln” function calculates the logarithm of a number of type LREAL to the base e (natural logarithm). The result is of type LREAL. If the input value is ≤ 0, a mathematical error is generated. This error can be determined with the “geterror_sr” function.

Block header:

```
Name:          ln
Type:          STANDARD_FUN

VAR_INPUT
  in           LREAL
VAR_END

VAR_OUTPUT
  ln           LREAL
VAR_END
```

FBD representation



Programming example:

```
ld      10.45
ln
```

Comment:

Load value into CR
Function call

7.1.1.12 log, Common logarithm

Description:

The “log” function calculates the logarithm of a number of type LREAL to the base 10 (common logarithm). The result is of type LREAL. If the input value is ≤ 0, a mathematical error is generated. This error can be determined with the “geterror_sr” function.

Block header:

```
Name:          log
Type:          STANDARD_FUN

VAR_INPUT
  in           LREAL
VAR_END

VAR_OUTPUT
  log          LREAL
VAR_END
```

FBD representation

Programming example:

```
ld      4.37E4
log
```

Comment:

Load value into CR
Function call

7.1.1.13 lreal_to_dint, Type conversion from LREAL to DINT

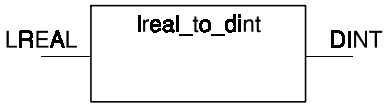
Description:

The “lreal_to_dint” function converts the data type LREAL to DINT. Decimal places are truncated. If the range of values is exceeded, the upper or lower DINT limit value is returned.

Block header:

Name:	lreal_to_dint
Type:	STANDARD_FUN
VAR_INPUT	
lreal_in	LREAL
VAR_END	
VAR_OUTPUT	
lreal_to_dint	DINT
VAR_END	

FBD representation



Programming example:		***Comment:***
ld	1.45E2	Load value into CR
lreal_to_dint		Function call

7.1.1.14 rol, Rotate CR to the left

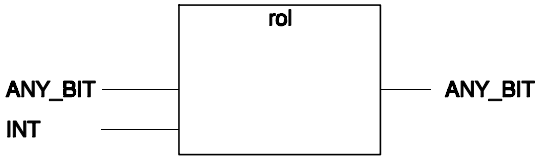
Description:

The “rol” function moves the contents of the CR by “n” bit positions to the left, rotating the bit values. The value in the CR must be of data type BOOL, BYTE or WORD.

Block header:

Name:	rol
Type:	STANDARD_FUN
VAR_INPUT	
in	ANY_BIT
n	INT
VAR_END	
VAR_OUTPUT	
rol	ANY_BIT
VAR_END	

FBD representation



Programming example:		***Comment:***
ld	2 # 1001 0111	Load value into CR
rol	3	Rotate the CR contents by 3 bits to the left (CR = 1011 1100)

7.1.1.15 ror, Rotate CR to the right

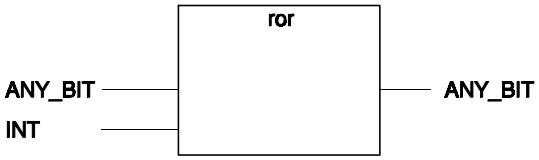
Description:

The “ror” function moves the contents of the CR by “n” bit positions to the right, rotating the bit values. The value in the CR must be of data type BOOL, BYTE or WORD.

Block header:

```
Name:          ror
Type:          STANDARD_FUN

VAR_INPUT
  in           ANY_BIT
  n            INT
VAR_END

VAR_OUTPUT
  ror          ANY_BIT
VAR_END
```

FBD representation



Programming example:

```
ld             2 # 1001 0111
ror            3
```

Comment:

Load value into CR
Rotate the CR contents by 3 bits to the right
(CR = 1111 0010)

7.1.1.16 shl, Shift CR to the left

Description:

The “shl” function moves the contents of the CR by “n” bit positions to the left, padding the CR on the right with zeros. The value in the CR must be of data type BOOL, BYTE or WORD.

Block header:

```
Name:          shl
Type:          STANDARD_FUN

VAR_INPUT
  in           ANY_BIT
  n            INT
VAR_END

VAR_OUTPUT
  shl          ANY_BIT
VAR_END
```

FBD representation

Programming example:

```
ld             2 # 1001 0111
shl            3
```

Comment:

Load value into CR
Shift the CR contents by 3 bits to the left
(CR = 1011 1000)

7.1.1.17 shr, Shift CR to the right

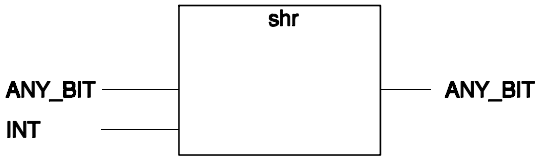
Description:

The “shr” function moves the contents of the CR by “n” bit positions to the right, padding the CR on the left with zeros. The value in the CR must be of data type BOOL, BYTE or WORD.

Block header:

```
Name:          shr
Type:          STANDARD_FUN

VAR_INPUT
  in           ANY_BIT
  n            INT
VAR_END

VAR_OUTPUT
  shr          ANY_BIT
VAR_END
```

FBD representation



Programming example:

```
ld      2 # 1100 1110
shr     4
```

Comment:

Load value into CR
Shift the CR contents by 4 bits to the right
(CR = 0000 1100)

7.1.1.18 sin, Sine function

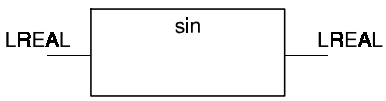
Description:

The “sin” function calculates the sine of an angle expressed in radians (rad). The valid range for the input value is $\pm 9.46 \times 10^{-308}$ to $\pm 1.79 \times 10^{+308}$. The data type of the result is LREAL. If the input value is not within the valid range, a mathematical error is generated. This error can be determined with the “geterror_sr” function.

Block header:

```
Name:          sin
Type:          STANDARD_FUN

VAR_INPUT
  in           LREAL
VAR_END

VAR_OUTPUT
  sin          LREAL
VAR_END
```

FBD representation



Programming example:

```
ld      1.25
sin
```

Comment:

Load radian angle into CR
Function call

7.1.1.19 sqrt, Calculation of square root

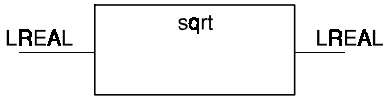
Description:

The “sqrt” function calculates the square root of a number of type LREAL. If the input value is < 0, a mathematical error is generated. This error can be determined with the “geterror_sr” function.

Block header:

```
Name:          sqrt
Type:          STANDARD_FUN

VAR_INPUT
  in           LREAL
VAR_END

VAR_OUTPUT
  sqrt         LREAL
VAR_END
```

FBD representation



Programming example:

```
ld             144.0
sqrt
```

Comment:

Load value into CR
Function call

7.1.1.20 tan, Tangent function

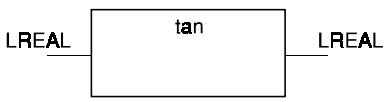
Description:

The “tan” function calculates the tangent of an angle expressed in radians (rad). The valid range for the input value is $\pm 9.46 \times 10^{-308}$ to $\pm 1.79 \times 10^{+308}$. The data type of the result is LREAL. If the input value is not within the valid range, a mathematical error is generated. This error can be determined with the “geterror_sr” function.

Block header:

```
Name:          tan
Type:          STANDARD_FUN

VAR_INPUT
  in           LREAL
VAR_END

VAR_OUTPUT
  tan          LREAL
VAR_END
```

FBD representation



Programming example:

```
ld             1.2
tan
```

Comment:

Load radian angle into CR
Function call

7.1.1.21 time_to_dint, Type conversion from TIME to DINT

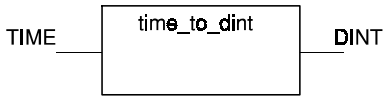
Description:

The “time_to_dint” function converts the data type TIME to DINT.

Block header:

```
Name:           time_to_dint
Type:           STANDARD_FUN

VAR_INPUT
  time          TIME
VAR_END

VAR_OUTPUT
  time_to_dint  DINT
VAR_END
```

FBD representation



Programming example:

```
ld              T# 3s 5ms
time_to_dint
```

Comment:

Load time into CR
Function call (result = 3005)

7.1.1.22 time_to_lreal, Type conversion from TIME to LREAL

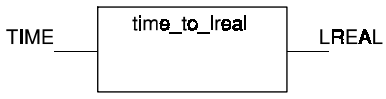
Description:

The “time_to_lreal” function converts the data type TIME to LREAL.

Block header:

```
Name:           time_to_lreal
Type:           STANDARD_FUN

VAR_INPUT
  time          TIME
VAR_END

VAR_OUTPUT
  time_to_lreal LREAL
VAR_END
```

FBD representation



Programming example:

```
ld              T#100ms
time_to_lreal
```

Comment:

Load time into CR
Function call

7.1.1.23 trunc, Rounding off numbers of type LREAL

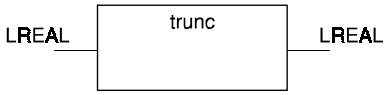
Description:

The “trunc” function truncates the decimal places of a number of type LREAL. The preceding sign is not affected.

Block header:

```
Name:          trunc
Type:          STANDARD_FUN

VAR_INPUT
  lreal        LREAL
VAR_END

VAR_OUTPUT
  trunc        LREAL
VAR_END
```

FBD representation



Programming example:

```
ld             1.83
trunc
```

Comment:

Load value 1.83 into CR
After the function call, 1 is the result stored in the CR

7.1.1.24 word_to_dint, Type conversion from WORD to DINT

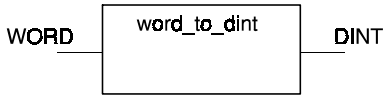
Description:

The “word_to_dint” function converts the data type WORD to DINT.

Block header:

```
Name:          word_to_dint
Type:          STANDARD_FUN

VAR_INPUT
  word         WORD
VAR_END

VAR_OUTPUT
  word_to_dint INT
VAR_END
```

FBD representation



Programming example:

```
ld             16#A4C9
word_to_dint
```

Comment:

Load data of type WORD into CR
Function call

7.1.2 Standard function blocks

The standard function blocks can be called from program blocks, global block and other function blocks. They include input/output variables for parameter passing.

The standard function blocks must be declared in the block headers of the calling blocks. This involves assigning a name to a function block type and allocating memory space for variables. This allows you to use several function blocks of the same type but with different names and allocated memory within a program block.

The following table contains all the standard function blocks, structured according to their uses:

Used for	**FB name**	**Meaning**	**See page**
Bistable elements	sr	Set/reset flipflop	7-27
	rs	Reset/set flipflop	7-25
Edge detection	r_trig	Trigger for leading edges	7-24
	f_trig	Trigger for trailing edges	7-23
Counters	ctd	Decremental counter	7-20
	ctu	Incremental counter	7-21
	ctud	Incremental/decremental counter	7-22
Timer functions	tp	Pulse timer	7-30
	ton	Timer ON delay	7-29
	tof	Timer OFF delay	7-28
Semaphore (global block)	sema	Semaphore	7-26

The following tables contain the corresponding function blocks with a description, block header, programming example with declarations and, if applicable, timing diagram.

The description is a brief explanation of the function.

In the block header, the name, the type and the input/output variables are defined.

The programming example illustrates block declaration and IL programming.

The FBD representation shows the function blocks in a graphic format.

NOTE
When calling a function block, not all input parameters must be specified since a function block has storage capability.

7.1.2.1 ctd, Decremental counter

Description:

The “ctd” function block is a decremental counter. It counts down in single increments (steps) from a definable start value to 0. The current counter value cv is decremented by 1 each time a positive edge is present on the counter input cd (in the previous function call, cd must have been FALSE). If the current counter value cv ≤ 0, the output q is set to TRUE.

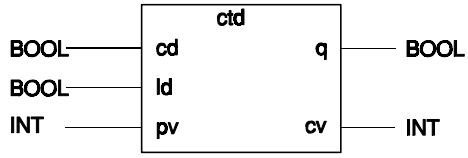
The input and output parameters have the following meanings:

cd	Counter input	Positive edge on cd ⟹ cv is decremented by 1.
ld	Load	If ld = TRUE, the start value pv is loaded into the current counter value cv.
pv	Start value	pv specifies the start value. The start value equals the number of steps by which ctd decrements until the output q is set.
q	Output	Is set to TRUE as soon as the current counter value cv ≤ 0.
cv	Current counter value	Is decremented by 1 if a positive edge is present on cd.

Block header:

```
Name:           ctd
Type:           STANDARD_FB

VAR_INPUT
  cd            BOOL
  ld            BOOL
  pv            INT
VAR_END

VAR_OUTPUT
  q             BOOL
  cv            INT
VAR_END

VAR
  cd_old        BOOL
VAR_END
```

FBD representation



Programming example:

```
VAR
  counter2      ctd
  pulse         BOOL
VAR_END

cal             counter2 (ld:=%IX0.1, pv:=200)

cal             counter2 (cd:=pulse)
ld              counter2.q
jmpc            END_COUNTER
```

Comment:

Declaration of a function block of type ctd.
Declaration of the local variable "pulse".

Function call with parameter passing. If input %IX0.1 is TRUE, the start value is set to 200.

Positive edge on cd ⟹ cv is decremented by 1.
Load value from output q into CR.
If CR = TRUE, jump to END_COUNTER label.

7.1.2.2 ctu, Incremental counter

Description:

The “ctu” function block is an incremental counter. It counts up in single increments (steps) from 0 to a definable limit value. The current counter value cv is incremented by 1 each time a positive edge is present on the counter input cu (cu = TRUE, however, in the previous function call, cu must have been FALSE). If the current counter value cv ≥ pv, the output q is set to TRUE.

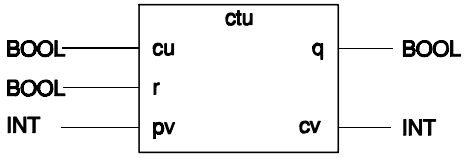
The input and output parameters have the following meanings:

cu	Counter input	Positive edge on cu ⟹ cv is incremented by 1.
r	Reset	If r = TRUE, cv is set to 0.
pv	Limit value	pv specifies the limit value. As soon as the current counter value cv ≥ pv, the output parameter q is set to TRUE.
q	Output	Is set to TRUE as soon as the current counter value cv ≥ pv.
cv	Current counter value	Is incremented by 1 when a positive edge is present on cu.

Block header:

```
Name:           ctu
Type:           STANDARD_FB

VAR_INPUT
  cu            BOOL
  r             BOOL
  pv            INT
VAR_END

VAR_OUTPUT
  q             BOOL
  cv            INT
VAR_END

VAR
  cu_old        BOOL
VAR_END
```

FBD representation



Programming example:

```
VAR
  counter1      ctu
  pulse         BOOL
VAR_END

cal             counter1 (r:=%IX0.2; pv:=100)
cal             counter1 (cu:=pulse)
ld              counter1.q
jmpc            END_COUNTER
```

Comment:

Declaration of a function block of type ctu.
Declaration of the local variable "pulse".

If input %IX0.2 = TRUE, cv is set to 0. The start value pv is 100.
Positive edge on cu ⟹ cv is incremented by 1.
Load value from output q into CR.
If CR = TRUE, jump to END_COUNTER label.

7.1.2.3 ctud, Incremental/decremental counter

Description:

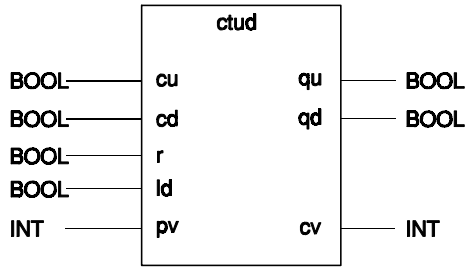
The “ctud” function block combines an incremental and a decremental counter (see also chapters 7.1.2.1 and 7.1.2.2). The input and output parameters have the following meanings:

cd	Counter input “down”	Positive edge on cd ⟹ cv is decremented by 1.
cu	Counter input “up”	Positive edge on cd ⟹ cv is incremented by 1.
r	Reset	If r = TRUE, cv is set to 0.
cv	Current counter value	Is incremented by 1 when a positive edge is present on cu. Is decremented by 1 if a positive edge is present on cd.
ld	Load	If ld = TRUE, the start/limit value pv is loaded into the current counter value cv.
pv	Start/limit value	pv specifies the start or limit value. When counting down, pv is the start value. When counting up, pv is the limit value.
qd	Output “down”	Is set to TRUE as soon as the current counter value cv ≤ 0.
qu	Output “up”	Is set to TRUE as soon as the current counter value cv ≥ pv.

Block header:

```
Name:              ctud
Type:              STANDARD_FB

VAR_INPUT
  cu               BOOL
  cd               BOOL
  r                BOOL
  ld               BOOL
  pv               INT
VAR_END

VAR_OUTPUT
  qu               BOOL
  qd               BOOL
  cv               INT
VAR_END

VAR
  cu_old           BOOL
  cd_old           BOOL
VAR_END
```

FBD representation



Programming example:

```
VAR
  counter3         ctud
  pulse            BOOL
VAR_END

cal                counter3 (r:=%IX0.1; pv:=100)
cal                counter3 (cu:=pulse)
ld                 counter3.qu
jmpc               END_COUNTER
```

Comment:

Declaration of a function block of type ctud.
Declaration of the local variable "pulse".

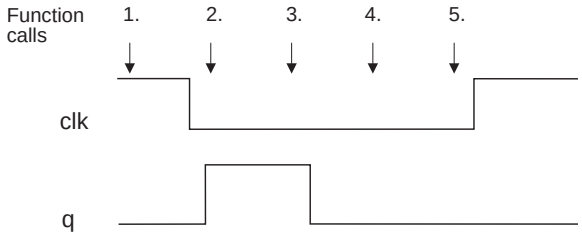
If input %IX0.1 = TRUE, cv is set to 0 and the limit value pv is loaded.
Positive edge on pulse ⟹ cv is incremented by 1.
Load output qu into CR.
If CR = TRUE, jump to END_COUNTER label.

7.1.2.4 f_trig, Trigger for trailing edges

Description:

The “f_trig” function block detects trailing edges. The value of the output q is 0 when the clk input changes from 1 to 0.

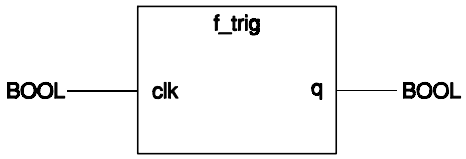
Timing diagram for f_trig



Block header:

```
Name:           f_trig
Type:           STANDARD_FB

VAR_INPUT
  clk           BOOL
VAR_END

VAR_OUTPUT
  q             BOOL
VAR_END

VAR
  clk_old       BOOL
VAR_END
```

FBD representation



Programming example:

```
VAR
  edge          f_trig
  pulse         BOOL
VAR_END

cal             edge(clk:=%IX0.1)
ld              edge.q
jmpc            EDGE_FALL
```

Comment:

Declaration of a function block of type f_trig.
Declaration of a local variable.

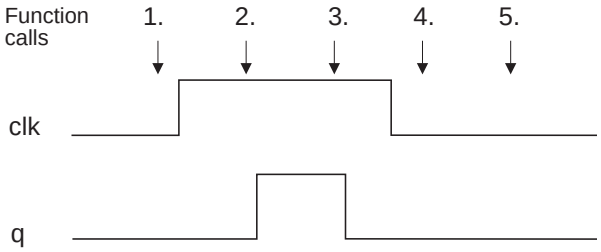
Function call with parameter passing.
Load output q into CR.
If CR = TRUE, jump to EDGE_FALL label.

7.1.2.5 r_trig, Trigger for leading edges

Description:

The “r_trig” function block detects leading edges. The value of the output q is 1 when the clk input changes from 0 to 1.

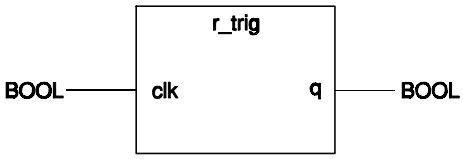
Timing diagram for r_trig



Block header:

```
Name:           r_trig
Type:           STANDARD_FB

VAR_INPUT
  clk           BOOL
VAR_END

VAR_OUTPUT
  q             BOOL
VAR_END

VAR
  m             BOOL
VAR_END
```

FBD representation



Programming example:

```
VAR
  edge          r_trig
  pulse         BOOL
VAR_END

cal             edge(clk:=%IX0.1)
ld              edge.q
st              pulse
```

Comment:

Declaration of a function block of type r_trig.
Declaration of a local variable.

Function call with parameter passing.
Load output q into CR.
The result is stored as “pulse”.

7.1.2.6 rs, Reset/set flipflop

Description:

The “rs” function block is a flipflop block. It has an output which is set to TRUE or FALSE, depending on the “set” and “reset” inputs. The “reset” input has priority.

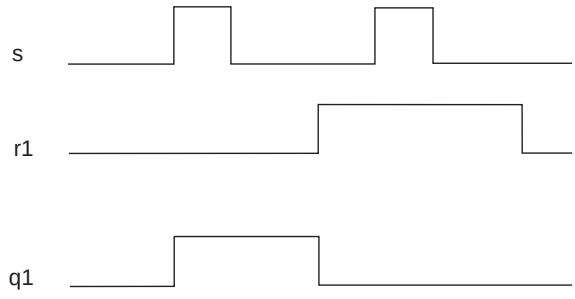
s = TRUE and r1 = FALSE $\Longrightarrow$ q1 = TRUE

s = FALSE and r1 = TRUE $\Longrightarrow$ q1 = FALSE

s = TRUE and r1 = TRUE $\Longrightarrow$ q1 = FALSE

The last condition shows that the “reset” input has priority.

Timing diagram for rs flipflop

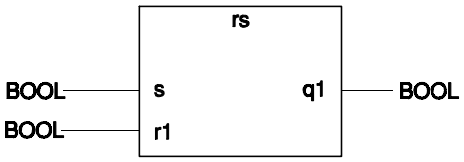


Block header:

Name: rs
Type: STANDARD_FB

```
VAR_INPUT
  s          BOOL
  r1         BOOL
VAR_END

VAR_OUTPUT
  q1         BOOL
VAR_END
```

FBD representation



Programming example:

```
VAR
  FF_29      rs
VAR_END

cal          FF_29 (s:=%IX0.1, r1:=%IX0.2)
ld           FF_29.q1
st           %QX0.1
```

Comment:

Declaration of a function block of type rs.

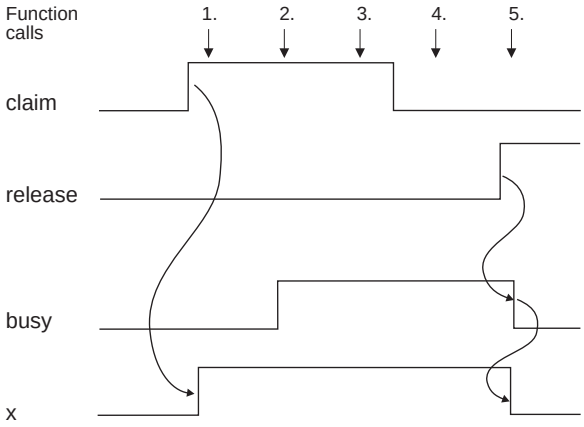
Function call with parameter passing.
Load output q1 into CR.
The output %QX0.1 is set or reset depending on the result.

7.1.2.7 sema, Semaphore (not yet implemented)

Description:

A semaphore is a task configuration flag which is used as a safety mechanism to prevent inadmissible (simultaneous) access (to certain data areas) when tasks are executed in parallel.

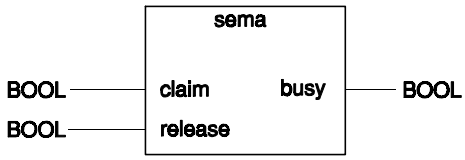
Timing diagram of semaphore



Block header:

Name:	sema
Type:	STANDARD_FB
VAR_INPUT	
claim	BOOL
release	BOOL
VAR_END	
VAR_OUTPUT	
busy	BOOL
VAR_END	
VAR	
x	BOOL
VAR_END	

FBD representation



7.1.2.8 sr, Set/reset flipflop

Description:

The “sr” function block is a flipflop block. It has an output which is set to TRUE or FALSE, depending on the “set” and “reset” inputs. The “set” input has priority.

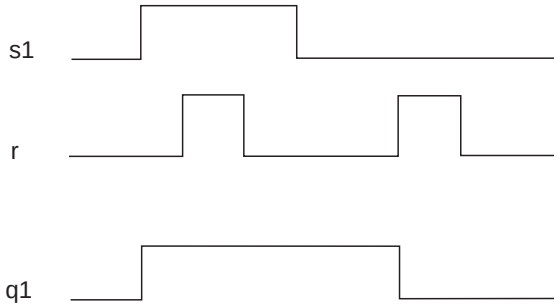
s1 = TRUE and r = FALSE $\Longrightarrow$ q1 = TRUE

s1 = FALSE and r = TRUE $\Longrightarrow$ q1 = FALSE

s1 = TRUE and r = TRUE $\Longrightarrow$ q1 = TRUE

The last condition shows that the “set” input has priority.

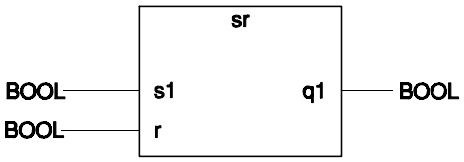
Timing diagram for sr flipflop



Block header:

Name:	sr
Type:	STANDARD_FB
VAR_INPUT	
s1	BOOL
r	BOOL
VAR_END	
VAR_OUTPUT	
q1	BOOL
VAR_END	

FBD representation



Programming example:

VAR		
FF_28	sr	Declaration of a function block of type sr.
VAR_END		
cal	FF_28 (s1:=%IX0.1, R:=%IX0.2)	Function call with parameter passing.
ld	FF_28.q1	Load output q1 into CR.
st	%QX0.1	The output %QX0.1 is set or reset depending on the result.

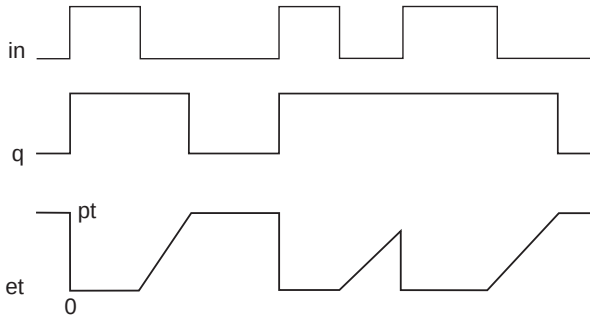
7.1.2.9 tof, Timer OFF delay

Description:

"tof" is a timer with timer OFF delay. A positive edge on input in sets the output q to TRUE. When the input in is set to FALSE, the timer starts. The output q continues to be TRUE. Only after the predefined time pt has elapsed, the output q is set to FALSE.

The current timeof the timer OFF delay can be retrieved from the output et.

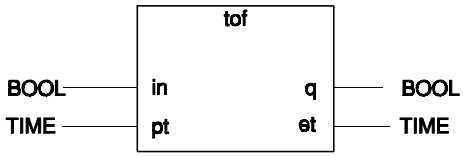
Timing diagram for timer OFF delay "tof"



Block header:

```
Name:           tof
Type:           STANDARD_FB

VAR_INPUT
  in            BOOL
  pt            TIME
VAR_END

VAR_OUTPUT
  q             BOOL
  et            TIME
VAR_END

VAR
  in_old        BOOL
  start_time    TIME
VAR_END
```

FBD representation



Programming example:

```
VAR
  offdel        tof
VAR_END

cal             offdel (in:=IX0.1, pt:=T#300 ms)
ld              offdel.q
st              %QX0.1
```

Comment:

Declaration of a function block of type tof.

Function call with parameter passing.
Load output q into CR.
The output %QX0.1 is set or reset depending on the result.

7.1.2.10 ton, Timer ON delay

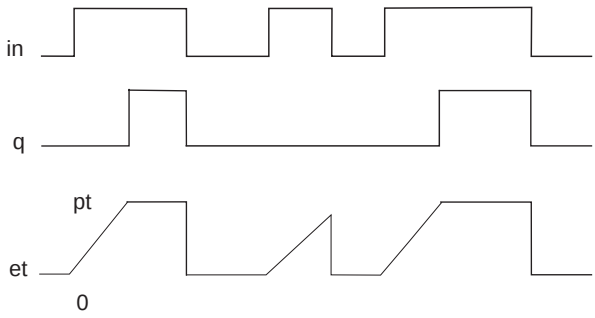
Description:

"ton" is a timer with timer ON delay. A positive edge on input in starts a timer. When the predefined time pt has elapsed, the output q is set to TRUE and retains this status until the input in is set to FALSE.

If the input signal is shorter than pt, output q remains FALSE.

The current time value of the timer ON delay can be retrieved from output et.

Timing diagram for timer ON delay "ton"

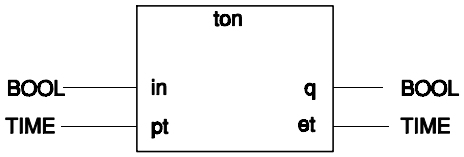


Block header:

Name: ton
Type: STANDARD_FB

```
VAR_INPUT
  in          BOOL
  pt          TIME
VAR_END

VAR_OUTPUT
  q           BOOL
  et          TIME
VAR_END

VAR
  in_old      BOOL
  start_time  TIME
VAR_END
```

FBD representation



Programming example:

```
VAR
  ondel       ton
VAR_END

cal           ondel (in:=%IX0.1, pt:=T#100 ms)
ld            ondel.q
st            %QX0.1
```

Comment:

Declaration of a function block of type ton.

Function call with parameter passing.
Load output q into CR.
The output %QX0.1 is set or reset depending on the result.

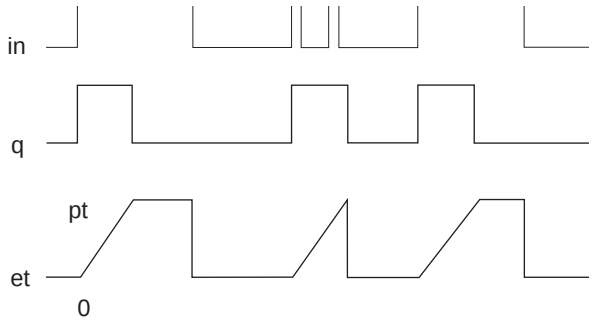
7.1.2.11 tp, Pulse timer

Description:

A positive edge on input in starts a timer and sets the output q to TRUE. The output q retains this status until the predefined time pt has elapsed. The status of the input in does not affect this condition.

The current time valueof the pulse timer can be retrieved anytime from output et.

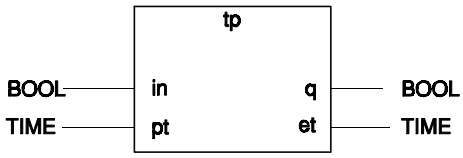
Timing diagram for pulse timer "tp"



Block header:

```
Name:           tp
Type:           STANDARD_FB

VAR_INPUT
  in            BOOL
  pt            TIME
VAR_END

VAR_OUTPUT
  q             BOOL
  et            TIME
VAR_END

VAR
  in_old        BOOL
  start_time    TIME
VAR_END
```

FBD representation



Programming example:

```
VAR
  pulse         tp
VAR_END

cal             pulse (in:=%IX0.1, pt:=T#100ms)
ld              pulse.q
st              %QX0.1
```

Comment:

Declaration of a function block of type tp.

Function call with parameter passing.
Load output q into CR.
The output %QX0.1 is set or reset depending on the result.

7.2 Controller library

The controller library of the programming system includes controller-specific functions. The function to be selected depends on the controller configuration specified in the programming system.

The following sections describe all the functions included in the controller library in a tabular format as follows:

Description: A brief description of the function	
Block header: Input/output parameters of the function  *NOTE* *For the VAR_IN_OUT parameter, a local variable (VAR) must be declared in the block header of the calling block.*	***FBD representation*** Symbol for FBD representation
Parameter(s): ***Value/constant:*** ***Explanation:*** Description of the input/output parameters with valid input values and system constants	
Programming example: Simple programming example for the function call  *NOTE* *All input parameters to the function must be specified with the function call in the order of their entries in the block header. The first input parameter is always the CR.*	
Function result: A description of the possible results of the function. The function result is stored in the CR after execution of the function.	

The following functions for controller programming are included in the controller library:

Controller function	Meaning	See page
accel	Select acceleration/deceleration curve	7-35
acclin	Calculate and activate linear acceleration/deceleration curve	7-36
brake	Define output for brake	7-37
calcaccel	Calculate acceleration/deceleration curve	7-38
clrerror_sr	Clear error condition	7-39
clrsig_sr	Clear temporarily stored axis signals, enable axis	7-40
	Clear temporarily stored encoder signals	7-41
com_init_asc	Initialize a serial interface for ASCII mode	7-42
continue	Continue interrupted axis movement	7-43
cycletime	Set PLC cycle time monitoring	7-44
debug	Reset outputs and stop axes at controller stop ON/OFF	7-45
display	Display number on controller's LED display	7-46
ensig	Enable or disable axis signals	7-47
getaccel	Get ramp assignment	7-48
getaccfac	Get maximum acceleration of a ramp	7-49
getanalog	Get analog value of an input channel from an analog module	7-50
getcurrent	Get electrical current values	7-51
getensig	Get enabled axis signals	7-52
geterror_sr	Get axis error condition	7-53
	Get encoder error condition	7-54
	Get linear interpolator error condition [2)]	7-55
	Get serial interface error condition	7-56
	Get mathematical error condition	7-57
gethardware	Get hardware signals	7-58
getmode	Get axis or encoder operating mode	7-59
getnorm	Get normalizing factors (axis)	7-60
	Get normalizing factors (encoder)	7-61
getparam [1)]	Get control parameters	7-62
getpos	Get axis positions in user-defined units	7-63
	Get encoder positions in user-defined units	7-64
getposd	Get axis positions in drive units	7-65
	Get encoder positions in encoder units	7-66
getsig	Get current axis signals	7-67
	Get current encoder signals	7-68
getsig_activ_h	Get active state of axis signals	7-69
getsig_sr	Get temporarily stored axis signals	7-70
	Get temporarily stored encoder signals	7-71

Controller function	Meaning	See page
getstate	Get axis status	7-72
	Get linear interpolation status conditions [2)]	7-75
	Get serial interface status	7-77
	Get battery status	7-78
	Get front panel key status	7-79
getvel	Get speed values	7-80
linmove2 [2)]	Relative linear interpolation with two axes	7-81
linmove2w [2)]	Relative linear interpolation with two axes and wait for end of movement	7-82
linmove3 [2)]	Relative linear interpolation with three axes	7-83
linmove3w [2)]	Relative linear interpolation with three axes and wait for end of movement	7-84
linpos2 [2)]	Absolute linear interpolation with two axes	7-85
linpos2w [2)]	Absolute linear interpolation with two axes and wait for end of movement	7-86
linpos3 [2)]	Absolute linear interpolation with three axes	7-87
linpos3w [2)]	Absolute linear interpolation with three axes and wait for end of movement	7-88
list	Disable position list processing	7-89
move	Relative positioning in user-defined units	7-90
moved	Relative positioning in drive units	7-91
movedf	Relative positioning in drive units and wait until position reached	7-92
movedw	Relative positioning in drive units and wait for end of movement	7-93
movef	Relative positioning in user-defined units and wait until position reached	7-94
movew	Relative positioning in user-defined units and wait for end of movement	7-95
pos	Absolute positioning in user-defined units	7-96
posd	Absolute positioning in drive units	7-97
posdf	Absolute positioning in drive units and wait until position reached	7-98
posdw	Absolute positioning in drive units and wait for end of movement	7-99
posf	Absolute positioning in user-defined units and wait until position reached	7-100
posw	Absolute positioning in user-defined units and wait for end of movement	7-101
receive_char	Read single character from transmit and receive buffer	7-102
receive_string	Read string from transmit and receive buffer	7-103
record	Start recording	7-104
refpos	Reference movement	7-105
refposf	Reference movement and wait until reference point reached	7-106
refposw	Reference movement and wait for end of movement	7-107
send_char	Output single characters to serial interface	7-108
send_string	Output string to serial interface	7-109
setaccsig	Signal-dependent deceleration	7-110
setanalog	Write analog value to an input channel of an analog module	7-111
setcurrent	Set electrical current values for various movement states	7-112

Controller function	Meaning	See page
setdrive	Set encoder input for position feedback	7-113
sethardware	Hardware settings for a serial interface	7-114
	Hardware settings for the axes	7-115
setmode	Set axis operating mode	7-116
	Set encoder operating mode	7-117
setnorm	Set normalizing factors for axis	7-118
	Set normalizing factors for encoder	7-119
setnormlist	Enable position list for gear ratios	7-120
setparam [1)]	Set control parameters	7-121
setpos	Set axis positions in user-defined units	7-122
	Set encoder positions in user-defined units	7-123
setposd	Set axis positions in drive units	7-124
	Set encoder positions in encoder units	7-125
setsig_activ_h	Set active state of axis signals	7-126
setsiglist	Enable position list for signal output	7-127
setsrcres	Set encoder input for position following mode	7-128
setsrctime	Set sampling time for position following mode	7-129
setsrctyp	Set reference variable type for position following mode	7-130
setsrcvar	Set variable for position following mode	7-131
setvel	Set start and maximum speeds	7-132
setvellist	Enable position list for speeds	7-133
sf_dint	Convert DINT to STRING and insert into string	7-134
sf_lreal	Convert LREAL to STRING and insert into string	7-136
sf_string	Insert or replace text	7-138
si_comp	Compare text	7-139
si_dint	Convert number string to DINT type	7-140
si_lreal	Convert number string to LREAL type	7-141
si_string	Extract substring	7-142
stop	Stop axis movement, stop controller	7-143
vel	Set the set speed	7-144
wait_time	Wait command for SEQUENCE programs	7-145

1) This function is only available on units with position controller: e.g. WDP3-337 and WDP3-338

2) This function is only available on multi-axis systems: e.g. WPM-311

7.2.1 accel, Select acceleration/deceleration curve

Description:

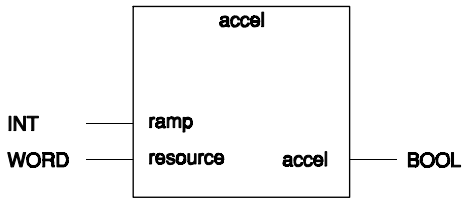
The manufacturer-defined function “accel” is used for selecting a ramp. This ramp is used for accelerating and decelerating the axis in the subsequent movements.

NOTE
The following requirements must be met prior to selecting a new ramp:
- *The axis must be at a standstill or inactive.*
- *The storage number of the ramp must be in the range from 1 to 10 (for acceleration values, see the controller manual).*
- *The ramp stored under the specified number must have been calculated with the manufacturer-defined function “calcaccel”.*
- *In speed mode and position following mode, “accel” may also be called during an axis movement.*

Block header:

```
Name:          accel
Type:          CONTROLLER_FUN

VAR_INPUT
  ramp         INT
  resource     WORD
VAR_END

VAR_OUTPUT
  accel        BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
ramp:	1-10	Storage numbers which reference the ramps (default: ramp 1)
resource:	x1, x2 ... xn	Axis 1, 2 ... n

Programming example:

```
ld          2
accel       x1
```

Comment:

Select ramp no. 2 as the active ramp

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.2 acclin, Calculate and activate linear acceleration/deceleration curve

Description:

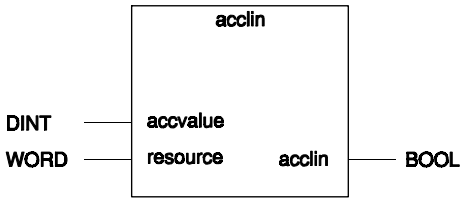
The “acclin” function calculates and activates a linear acceleration/deceleration curve. The currently active curve is overwritten.

NOTE
In point-to-point mode, “acclin” may only be called when the axis is at a standstill.

Block header:

```
Name:          acclin
Type:          CONTROLLER_FUN

VAR_INPUT
  accvalue     DINT
  resource     WORD
VAR_END

VAR_OUTPUT
  acclin       BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
accvalue:	Acceleration value	Acceleration in user-defined units
resource:	x1, x2 ... xn	Axis 1, 2 ... n

Programming example:

```
ld      200
acclin  x1
```

Comment:

Specify 200 user-defined units for the linear acceleration for axis 1

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.3 brake, Define output for brake

Description:

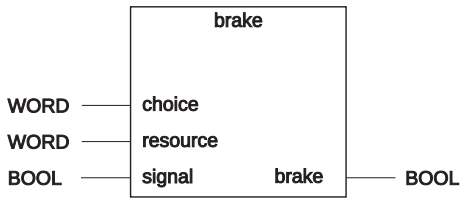
The manufacturer-defined function “brake” is used for linking any controller output to the active state of the power amplifier. This output can then be used directly for controlling a brake.

ATTENTION
An output with this assignment can still be modified via the process image or by setting/resetting it directly from an application program. When debugging, you should take care not to reset the outputs (by breakpoints) in case of a controller stop. You can set this using the “debug” function.
Unplugging the motor connector on the unit is not recognized and fed back to the controller by resetting the ready signal by all power amplifiers.

Block header:

```
Name:          brake
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
VAR_END

VAR_OUTPUT
  brake        BOOL
VAR_END


VAR_IN_OUT
  signal       BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	brake1	Brake selection parameter
resource:	x1, x2 ... xn	Axis 1, 2 ... n
signal:		Output designation

Programming example:

```
ld             brake1
brake          x1, %QX0.3
```

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.4 calcaccel, Calculate acceleration/deceleration curve

Description:

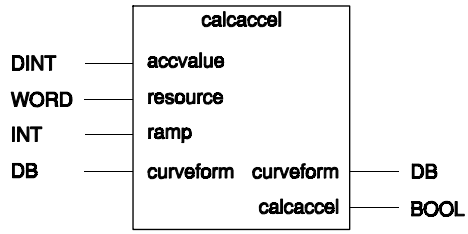
The manufacturer-defined function “calcaccel” is used for calculating a ramp from the specified master curve and the maximum acceleration and saving it under the specified storage number (1-10). Acceleration curves may only be calculated while the axis is at a standstill.

NOTE
For more detailed information on calculating acceleration/deceleration curves, see chapter 3.3.

Block header:

```
Name:          calcaccel
Type:          CONTROLLER_FUN

VAR_INPUT
  accvalue     DINT
  resource     WORD
  ramp         INT
VAR_END

VAR_OUTPUT
  calcaccel    BOOL
VAR_END

VAR_IN_OUT
  curveform    ANY_DB
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
accvalue:	Acceleration value	Maximum acceleration in user-defined units
resource:	x1, x2 ... xn	Axis 1, 2 ... n
ramp:	1 - 10	Storage number used for saving a ramp
curveform:	Data block name	Data block with master curve

Programming example:		***Comment:***
ld	500	The "curveform" master curve is normalized to the value 500 and stored as ramp no. 2.
calcaccel	x1, 2, curveform 1	

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE). The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19. An error is detected if:

- the maximum acceleration value exceeds the limit
- the maximum acceleration value is less than or equal to 0
- the storage number is not within the valid range (1-10)
- the master curve has an invalid shape.

7.2.5 clrerror_sr, Clear error condition

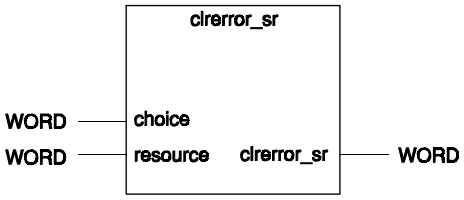
Description:

The manufacturer-defined function “clrerror_sr” can be used to clear error conditions (see “geterror_sr”). All error bits (except “err_watch”) are cleared in the corresponding error word. The function also clears any error codes displayed on the controller display.

Block header:

```
Name:          clrerror_sr
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
VAR_END

VAR_OUTPUT
  clrerror_sr  BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	err_all	All error conditions
resource:	x1, x2 ... xn	Axis 1, 2 ... n
	p1, p2	Encoder 1, 2
	c1, c2	Serial interface 1, 2
	mathe	Mathematical error
	l1 [1)]	Linear interpolator

Programming example:

```
ld          err_all
clrerror_sr p1
```

Comment:

Clear all error bits (except “err_watch”)
Resource: encoder 1

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).

NOTE
For information on how to retrieve error conditions using “geterror_sr”, see chapters 7.2.19, 7.2.20, 7.2.21, 7.2.22 and 7.2.23.

1) Only available on multi-axis systems: e.g. WPM-311

7.2.6 clrsig_sr, Clear temporarily stored axis signals, enable axis

Description:

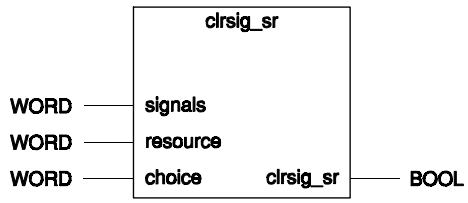
The manufacturer-defined function “clrsig_sr” can be used to clear temporarily stored axis signals (signal flags). If the axis movement was stopped by a signal, the axis can be re-enabled after having come to a standstill, using “clrsig_sr”. The “trig” signal may also be reset during axis movement.

NOTE
- *A flag can only be successfully reset if the trigger signal is no longer active.*
- *“clrsig_sr” must never be invoked during axis movement.*

Block header:

```
Name:          clrsig_sr
Type:          CONTROLLER_FUN

VAR_INPUT
  signals      WORD
  resource     WORD
  choice       WORD
VAR_END

VAR_OUTPUT
  clrsig_sr    BOOL
VAR_END
```

FBD representation



Parameter:	***Value/constant:***	***Description:***
signals:		Clearing the signal flags:
	limp	Positive hardware limit switch
	limn	Negative hardware limit switch
	ref	Reference switch
	stop	Hardware stop input
	trig	Hardware trigger
	swlimp	Positive software limit switch
	swlimn	Negative software limit switch
	swstop	Software stop
	dragerr	Contouring error (if contouring error cleared on encoder; see chapter 7.2.7)
	encerr	Encoder error
	ampnotready	Power controller not ready
	amptemp	Power controller overtemperature
	motortemp	Motor overtemperature
	allsig	All signals
resource:	x1, x2 ... xn	Axis 1, 2 ... n
	(p1, p2)	(Encoder 1, encoder 2; see chapter 7.2.7)
choice:	watch	Monitoring signals

Programming example:		***Comment:***
ld	limp	To reset several individual signal flags, the system constants must be specified with an OR operation.
or	limn	
or	dragerr	
clrsig_sr	x1, watch	

Function result:
If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE). The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.7 clrsig_sr, Clear temporarily stored encoder signals

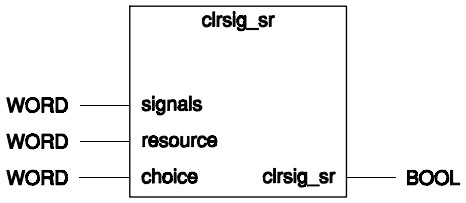
Description:

The manufacturer-defined function “clrsig_sr” can be used to clear temporarily stored encoder signals (signal flags). The signal flag of a contouring error must be cleared before it is possible to release the axis from its blocked condition. Of course, first the rotation monitoring error must be eliminated either by disabling rotation monitoring or by setting the following error to a value which is less than the defined contouring error limit value.

Block header:

```
Name:          clrsig_sr
Type:          CONTROLLER_FUN

VAR_INPUT
  signals      WORD
  resource     WORD
  choice       WORD
VAR_END

VAR_OUTPUT
  clrsig_sr    BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
signals:		Clearing the signal flags:
	dragerr	Contouring error
	encerr	Encoder error
	index[1)]	Index pulse
resource:	p1, p2	Encoder 1, encoder 2
	(x1, x2 ... xn)	(Axis 1, 2 ... n; see chapter 7.2.6)
choice:	watch	Monitoring signals

Programming example:

```
ld             dragerr
clrsig_sr      p1, watch
```

Function result:

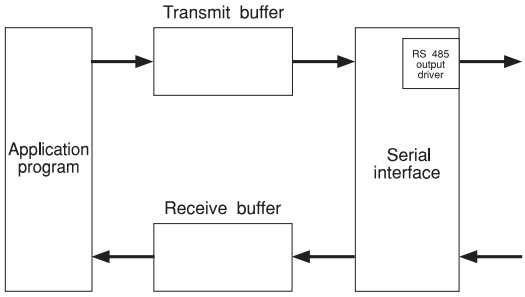
If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.20.

1) Only available for resource p2

7.2.8 com_init_asc, Initialize serial interface for ASCII mode

Description:
The manufacturer-defined function “com_init_asc” initializes a serial interface, which involves allocation of transmit and receive buffers and setting the interface parameters.

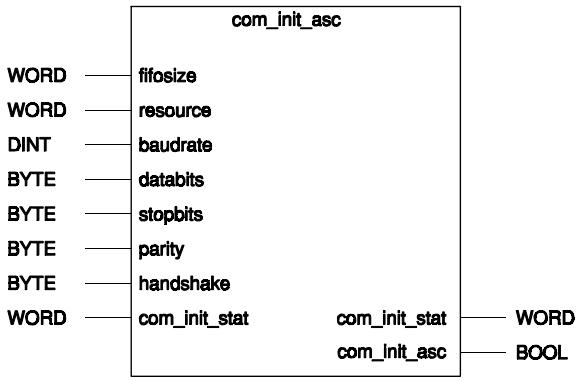
NOTE
Data can only be transmitted after the RS 485 output driver has been activated with “sethardware”.



Block header:

Name:	com_init_asc
Type:	CONTROLLER_FUN
VAR_INPUT	
fifosize	WORD
resource	WORD
baudrate	DINT
databits	BYTE
stopbits	BYTE
parity	BYTE
handshake	BYTE
VAR_END	
VAR_OUTPUT	
com_init_asc	BOOL
VAR_END	
VAR_IN_OUT	
com_init_stat	WORD
VAR_END	

FBD representation



Parameter(s):	***Value/constant:***	***Description:***
fifosize:	30 to 32767 characters	Buffer size for transmit and receive string
resource:	c1, c2	Serial interface 1, 2
baudrate:	[1]	Interface baud rate
databits:	6, 7, 8 bits	Number of data bits
stopbits:	1, 2 bits	Number of stop bits
parity:	no, even, odd	No, even or odd parity bit
handshake:	no, xon_xoff	No handshake, software handshake, hardware handshake
	break_on	Break detection (OR operation with handshake constant)
com_init_stat		Status word; see also “getstate” in chapter 7.2.38

Programming example:

```
VAR
  com_status        WORD
VAR_END

ld                  1024
com_init_asc        c2, 9600, 8, 1, even, xon_xoff, com_status
```

Function result:
If executed successfully, the function returns TRUE, otherwise FALSE. If the function returns FALSE, the cause can be determined from the status returned. You can also use “getstate” to retrieve the status explicitly from another point in the program and retrieve a potential error cause using “geterror_sr”.

1) 75, 110, 150, 300, 600, 1200, 1800, 2000, 2400, 4800, 9600, 19200, 38400 bauds

7.2.9 continue, Continue interrupted axis movement

Description:

The manufacturer-defined function “continue” can be used for resuming an interrupted axis movement. The effect of the function depends on the axis operating mode:

Point-to-point mode
The previously defined setpoint is re-set. If the axis movement is interrupted, or blocked, only the setpoint is re-set without performing a movement.

Speed mode
The previously defined set speed is re-set. If the axis movement is interrupted, or blocked, only the setpoint is re-set without performing a movement.

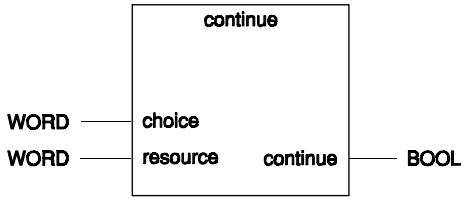
Position following mode
No effect.

NOTE
To execute the continue command, the following requirements must be met:
- *The axis is at a standstill, or inactive.*
- *The acceleration must be defined.*
- *No reference movement is active.*
- *The drive is not blocked.*

Block header:

```
Name:          continue
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
VAR_END

VAR_OUTPUT
  continue     BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	drive	Indexer (not significant)
resource:	x1, x2 ... xn	Axis 1, 2 ... n

Programming example:

```
ld          stop
clrsig_sr   x1, watch
ld          drive
continue    x1
```

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.10 cycletime, Set PLC cycle time monitoring

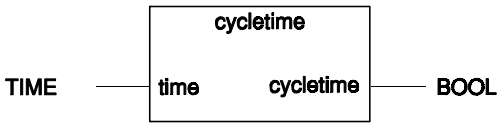
Description:

The “cycletime” function sets the time for the PLC cycle time monitoring feature. The PLC cycle time is the time the controller takes to execute the PLC task once (see figure 1-9). The PLC cycle time monitoring feature monitors the PLC cycle time. If the value preset with “cycletime” is exceeded, the controller stops. If you specify 0, the PLC cycle time monitoring feature is disabled.

Block header:

```
Name:          cycletime
Type:          CONTROLLER_FUN

VAR_INPUT
  time         TIME
VAR_END

VAR_OUTPUT
  cycletime    BOOL
VAR_END
```

FBD representation



Parameter(s):	***Explanation:***
time:	Maximum allowed time for one cycle.

Programming example:

```
ld             T#100 ms
cycletime
```

Function result:

This function always returns TRUE.

7.2.11 debug, Reset outputs and stop axes at controller stop ON/OFF

Description:

When the controller is stopped, two software switches control whether or not the outputs are reset and the axes stopped. These software switches are toggled with the manufacturer-defined function “debug”.

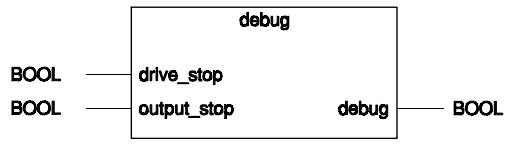
ATTENTION
By default, outputs and axes are reset automatically at program stop. This feature can be disabled with the “debug” function. However, the following effects must always be taken into account:
Outputs and axes are no longer monitored by the controller after a program stop. This means that, for example, motors continue to run without control, which may result in injury or damage!

Block header:

Name: debug
Type: CONTROLLER_FUN

```
VAR_INPUT
  drive_stop    BOOL
  output_stop   BOOL
VAR_END

VAR_OUTPUT
  debug         BOOL
VAR_END
```

FBD representation



Parameter(s):	***Explanation:***
drive_stop:	If FALSE is specified for this parameter, the axes are not stopped when the controller stops. TRUE reverses this condition.
output_stop:	If FALSE is specified for this parameter, the outputs are not reset when the controller stops. TRUE reverses this condition.

Programming example:

```
ld      FALSE
debug   FALSE
```

When invoking these calls, outputs and axes will not be reset or stopped, respectively, when the controller stops. TRUE reverses this condition.

Function result:

This function always returns TRUE.

7.2.12 display, Display number on controller LED display

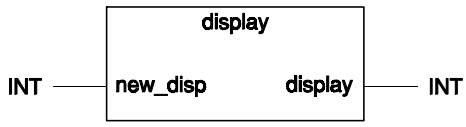
Description:

The “display” function outputs a numerical value on the two-digit LED display of the controller.

Block header:

```
Name:          display
Type:          CONTROLLER_FUN

VAR_INPUT
  new_disp     INT
VAR_END

VAR_OUTPUT
  display      INT
VAR_END
```

FBD representation



Parameter(s):	***Explanation:***
new_disp:	Numerical value to be displayed. If -1 is specified, the LED display is cleared.

Programming example:

```
ld             87
display
```

Function result:

The function returns the value displayed.
If no value was displayed, the function returns -1.

7.2.13 ensig, Enable/disable axis signals

Description:

The manufacturer-defined function “ensig” can be used for enabling or disabling axis signals. A signal is enabled by a "1" at the corresponding bit position, and it is disabled by a "0". Factory settings: “limp”, “limn”, “ref”, “step” and “ampnotready” are enabled, all other signals are disabled.

NOTE
- *Disabled signals do not affect the axis. Therefore, predefined signal inputs (e.g. “ref”) can be freely used. The software stop signal cannot be disabled.*
- *“ensig” must not be called during a reference movement.*
- *Disabled signals continue to be available for retrieval by “getsig_sr” and “getsig”.*

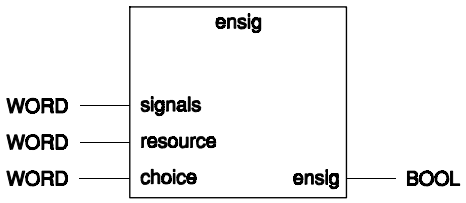
ATTENTION
When using WDP3-337 or WDP3-338 units the monitoring signals “ampnotready” and “dragerr” must not be disabled.

Block header:

Name: ensig
Type: CONTROLLER_FUN

```
VAR_INPUT
  signals     WORD
  resource    WORD
  choice      WORD
VAR_END

VAR_OUTPUT
  ensig       BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Description:***
signal:		Enabling/disabling the following signals:
	limp	Positive hardware limit switch
	limn	Negative hardware limit switch
	ref	Reference switch
	stop	Hardware stop input
	trig	Hardware trigger
	swlimp	Positive software limit switch (only for point-
	swlimn	Negative software limit switch to-point mode)
	dragerr	Contouring error
	encerr	Encoder error
	ampnotready	Power controller not ready
	amptemp	Power controller overtemperature
	motortemp	Motor overtemperature
	allsig	All signals
resource:	x1, x2 ... xn	Axis 1, 2 ... n
choice:	watch	Monitoring signals

Programming example:

```
ld          ref
or          trig
or          dragerr
ensig       x1, watch
```

Comment:
To enable several individual axis signals, the system constants must be specified with an OR operation.

Function result:
If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE). The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

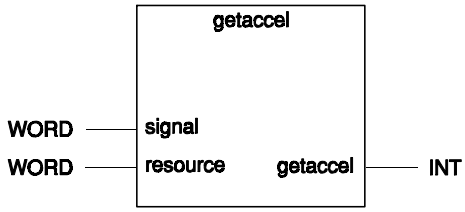
7.2.14 getaccel, Get ramp assignment

Description:
The manufacturer-defined function “getaccel” is used for retrieving ramp assignment information.

Block header:

```
Name:          getaccel
Type:          CONTROLLER_FUN

VAR_INPUT
  signal       WORD
  resource     WORD
VAR_END

VAR_OUTPUT
  getaccel     INT
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
signal:	limp	Ramp number activated upon encountering a positive hardware limit switch
	limn	Ramp number activated upon encountering a negative hardware limit switch
	ref	Ramp number activated upon encountering a reference switch
	stop	Ramp number activated upon encountering a hardware stop input
	swstop	Ramp number activated upon encountering a software stop
	dragerr	Ramp number activated upon encountering a contouring error
	actual	Ramp number of the currently loaded ramp
	selected	Ramp number selected with the manufacturer-defined function “accel”
resource:	x1, x2 ... xn	Axis 1, 2 ... n

Programming example:

```
ld          ref
getaccel    x1
```

Function result:

If executed successfully, the function returns the storage number of the corresponding ramp; if the function fails, it returns 0.

The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.15 getaccfac, Get maximum acceleration of a ramp

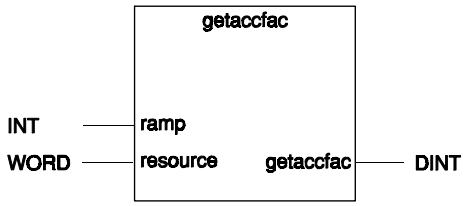
Description:

The manufacturer-defined function “getaccfac” returns the maximum acceleration on the basis of which the ramp was calculated. The maximum acceleration is specified in user-defined units.

Block header:

```
Name:          getaccfac
Type:          CONTROLLER_FUN

VAR_INPUT
  ramp         INT
  resource     WORD
VAR_END

VAR_OUTPUT
  getaccfac    DINT
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
ramp:	1 - 10	Storage number
resource:	x1, x2 ... xn	Axis 1, 2 ... n

Programming example:

```
ld          1
getaccfac   x1
```

Function result:

If executed successfully, the function returns the maximum acceleration value (in user-defined units) of the specified ramp.
If the function fails (e.g. ramp not defined), it returns 0.

The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.16 getanalog, Get analog value of an input channel from an analog module

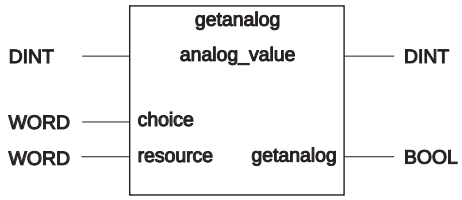
Description:

This function can be used to read the analog value of an input channel from an analog module.

Block header:

```
Name:           getanalog
Type:           CONTROLLER_FUN

VAR_INPUT
  choice        WORD
  resource      WORD
VAR_END

VAR_IN_OUT
  analog_value  DINT
VAR_END

VAR_OUTPUT
  getanalog     BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	1, 2, 3, 4, 5	Number of analog input on analog module
resource:	ad1	Analog module slot 51
	ad2	Analog module slot 53
	ad3	Encoder module with analog input in slot 55
analog_value:		Analog value in mV

Programming example:

Reading the second analog input on the analog module slot 53

```
VAR
  value         DINT
VAR_END

ld              2
getanalog       ad2, value
```

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
As a result, the analog value in mV is passed in “value”.

7.2.17 getcurrent, Get electrical current values

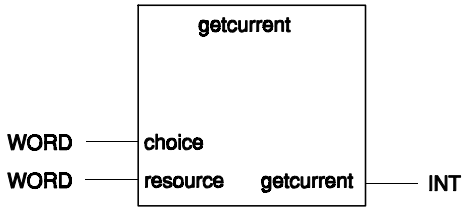
Description:

The manufacturer-defined function “getcurrent” can be used to determine the electrical current values (expressed as a percentage) set previously with the manufacturer-defined function “setcurrent”.

Block header:

```
Name:          getcurrent
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
VAR_END

VAR_OUTPUT
  getcurrent   INT
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	actual	Currently set electric current
	stand	Current at standstill
	accel	Current for acceleration and deceleration phase
	constant	Current for constant movement
resource:	x1, x2 ... xn	Axis 1, 2 ... n

Programming example:

```
ld             stand
getcurrent     x1
```

Function result:

If executed successfully, the function returns the electrical current value selected with the choice parameter (expressed as a percentage of the set phase current). If the function fails, it returns 0.

The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.18 getensig, Get enabled axis signals

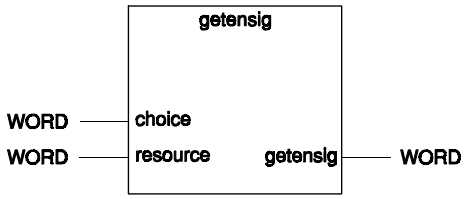
Description:

The manufacturer-defined function “getensig” can be used to retrieve the axis signals enabled or disabled with the manufacturer-defined function “ensig”. A "1" at the corresponding bit position means that a signal is enabled, a "0" means that it is disabled.

Block header:

```
Name:          getensig
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
VAR_END

VAR_OUTPUT
  getensig     WORD
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	watch	Monitoring signals
resource:	x1, x2 ... xn	Axis 1, 2 ... n

Programming example:		***Comment:***
ld	watch	Enable contouring error
getensig	x1	
or	dragerr	
ensig	x1, watch	

Function result:

If an error occurs during execution, the function returns 0 (FALSE). If executed successfully, the function returns a bit pattern which can be evaluated according to the above programming example using the following system constants:

limp	Positive hardware limit switch
limn	Negative hardware limit switch
ref	Reference switch
stop	Hardware stop input
trig	Hardware trigger
swlimp	Positive software limit switch
swlimn	Negative software limit switch
swstop	Software stop
dragerr	Contouring error
encerr	Encoder error
ampnotready	Power controller not ready
amptemp	Power controller overtemperature
motortemp	Motor overtemperature

If the function fails, it returns 0.
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.19 geterror_sr, Get axis error condition

Description:

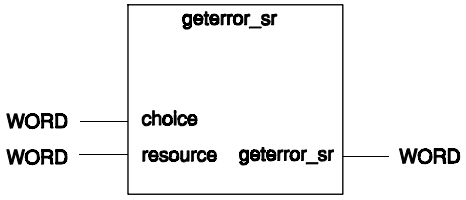
If an error occurs during a function call which addresses the axis (function result = FALSE), information on the type of the error is stored in an error word. The “geterror_sr” function can be used to determine the contents of this error word.

NOTE
The error bits can be cleared with the “clrerror_sr” function, except bit 0. Bit 0 (err_watch) is cleared as soon as all axis monitoring signals are inactive and reset with “clrsig_sr”.

Block header:

```
Name:           geterror_sr
Type:           CONTROLLER_FUN

VAR_INPUT
  choice        WORD
  resource      WORD
VAR_END
VAR_OUTPUT
  geterror_sr   WORD
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	err_all	General error information on the axis
resource:	x1, x2 ... xn	Axis 1, 2 ... n
	(p1, p2)	(Encoder 1, encoder 2; see chapter 7.2.20)
	(c1, c2)	(Serial interface 1, 2; see chapter 7.2.22)
	(mathe)	(Mathematical error; see chapter 7.2.23)
	(l1)	(Linear interpolator; see chapter 7.2.21)

Programming example:

```
ld              err_all
geterror_sr     x1
```

Function result:

If an error occurs during execution, the function returns 0 (FALSE). If executed successfully, the function returns a bit pattern which can be evaluated using the following system constants (for an evaluation example, see “getensig”).

err_watch	Monitoring signal set (active). The enabled monitoring signal can be determined using “getsig_sr”. Bit 0 is automatically reset when no monitoring signal is active.
err_w_extern	Invalid command in position following mode
err_wait_pos_activ	Positioning with wait status already active (e.g. posw, movew, posf, movef, linpos2w)
err_drive_stopped	Invalid command while axis is interrupted/blocked
err_acc_list	Invalid master curve
err_src_info	Insufficient information on the reference variable
err_drive_move	Invalid command while axis moving
err_ref_activ	Invalid command during reference movement
err_choice	Invalid parameter value
err_calc	A value cannot be calculated
err_parameter	Invalid value passed for a parameter
err_condition	Command cannot be executed under the circumstances
err_undef	Attempt to retrieve an undefined value
err_res_ready	Resource not ready
err_ipo_activ	Invalid command during interpolation process

7.2.20 geterror_sr, Get encoder error condition

Description:

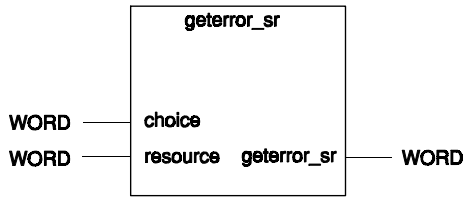
If an error occurs during a function call which addresses an encoder (function result = FALSE), information on the type of the error is stored in an error word. The “geterror_sr” function can be used to determine the contents of this error word. Each bit of the error word has a defined significance.

NOTE
The error bits can be cleared with the “clrerror_sr” function, except bit 0. Bit 0 (err_watch) is cleared as soon as all encoder monitoring signals are inactive.

Block header:

```
Name:          geterror_sr
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
VAR_END

VAR_OUTPUT
  geterror_sr  WORD
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	err_all	General error information
resource:	p1, p2	(Encoder 1, encoder 2
	(x1, x2 ... xn)	(Axis 1, 2 ... n; see chapter 7.2.19)
	(c1, c2)	(Serial interface 1, 2; see chapter 7.2.22)
	(mathe)	(Mathematical error; see chapter 7.2.23)
	(l1)	(Linear interpolator; see chapter 7.2.21)

Programming example:

```
ld             err_all
geterror_sr    p2
```

Function result:

If an error occurs during execution, the function returns 0 (FALSE). If executed successfully, the function returns a bit pattern which can be evaluated using the following system constants (for an evaluation example, see “getensig”).

err_watch	Monitoring signal set (active). The enabled monitoring signal can be determined using “getsig_sr”.
err_choice	Invalid parameter value
err_calc	A value cannot be calculated
err_parameter	Invalid value passed for a parameter
err_condition	Command cannot be executed under the circumstances

7.2.21 geterror_sr, Get linear interpolator error condition

Description:

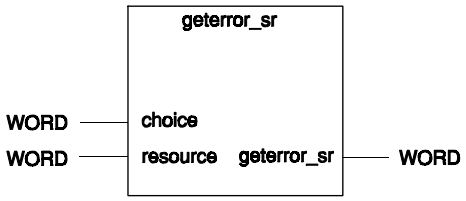
If an error occurs during a function call which addresses the linear interpolator resource or any axis involved in interpolation (function result = FALSE), information on the type of the error is stored in an error word. The error word of the linear interpolator is a result of the OR operation performed with all error words of the axes involved in interpolation. The “geterror_sr” function can be used to retrieve this error word.

NOTE
The error bits can be cleared with the “clrerror_sr” function, except bit 0. Bit 0 (err_watch) is cleared as soon as all monitoring signals of the axes involved in interpolation are inactive and reset with “clrsig_sr”.

Block header:

```
Name:          geterror_sr
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
VAR_END

VAR_OUTPUT
  geterror_sr  WORD
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	err_all	General error information on the axis
resource:	l1	Linear interpolator
	(x1, x2 ... xn)	(Axis 1, 2 ... n; see chapter 7.2.19)
	(p1, p2)	(Encoder 1, encoder 2; see chapter 7.2.20)
	(c1, c2)	(Serial interface 1, 2; see chapter 7.2.22)
	(mathe)	(Mathematical error; see chapter 7.2.23)

Programming example:

```
ld              err_all
geterror_sr     l1
```

Function result:

If an error occurs during execution, the function returns 0 (FALSE). If executed successfully, the function returns a bit pattern which can be evaluated using the following system constants (for an evaluation example, see “getensig”).

err_watch	Monitoring signal set (active) for one or more axes involved in interpolation. The enabled monitoring signal can be determined using “getsig_sr”.
err_wait_pos_activ	Positioning with wait status already active (e.g. posw, movew, posf, movef, linpos2w)
err_drive_stopped	Invalid command while axis is interrupted/blocked
err_choice	Invalid parameter value
err_calc	A value cannot be calculated
err_parameter	Invalid value passed for a parameter
err_condition	Command cannot be executed under the circumstances
err_undef	Attempt to retrieve an undefined value
err_res_ready	Resource not ready

7.2.22 geterror_sr, Get serial interface error condition

Description:
If an error occurs during a function call which addresses a serial interface (function result = FALSE), information on the type of the error is stored in an error word. The "geterror_sr" function can be used to determine the contents of this error word. Each bit of the error word has a defined significance.

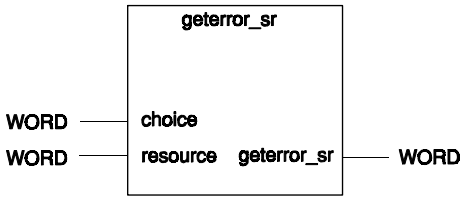
NOTE
The error bits can be cleared with the "clrerror_sr" function.

NOTE
Further options for error retrieval with regard to axis, encoder and mathematical errors are described in chapters 7.2.19, 7.2.20 and 7.2.23.

Block header:

```
Name:          geterror_sr
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
VAR_END

VAR_OUTPUT
  geterror_sr  WORD
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	err_all	General error information
	err_com	Special errors on the serial interface 1 or 2 resource
resource:	c1, c2	Serial interface 1, 2
	(x1, x2 ... xn)	(Axis 1, 2 ... n; see chapter 7.2.19)
	(p1, p2)	(Encoder 1, encoder 2; see chapter 7.2.20)
	(mathe)	(Mathematical error; see chapter 7.2.23)
	(l1)	(Linear interpolator; see chapter 7.2.21)

Programming example:

```
ld             err_com
geterror_sr    c2
```

Function result:
If an error occurs during execution, the function returns 0 (FALSE). If executed successfully, the function returns a bit pattern which can be evaluated using the following system constants:

CHOICE: err_all	
err_choice	Invalid parameter value
err_parameter	Invalid value passed for a parameter
CHOICE: err_com	
err_comnotinit	Attempt to access non-initialized interface
err_comoccupied	Hardware interface already assigned
err_memalloc	Error in allocating or disallocating buffer space
err_recbufsize	Receive buffer string too small
err_sendbufsize	Transmit string too long
err_sendbufdata	Invalid character string in transmit string
err_ibufoverrun	Receive buffer overflow
err_charoverrun	Overrun error: Baud rate may be set too high
err_parity	Parity error: Transmission error
err_framing	Framing error: Interface parameter set incorrectly
err_break	Break error: Interface cable interrupted, or no link available

7.2.23 geterror_sr, Get mathematical error condition

Description:

If an error occurs in a function call (function result = FALSE), you can use the “geterror_sr” function to check whether it is a mathematical error. Mathematical errors are:

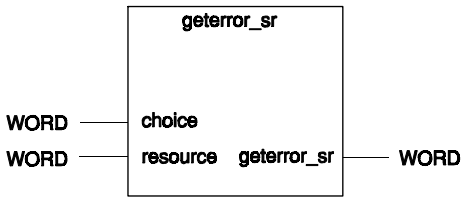
- Division by 0
- Invalid input value to a function
- Memory overflow while executing a function or performing a calculation

NOTE
The error bit can be cleared with the “clrerror_sr” function.

Block header:

```
Name:          geterror_sr
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
VAR_END

VAR_OUTPUT
  geterror_sr  WORD
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	err_all	General error information
resource:	mathe	Mathematical error
	(x1, x2 ... xn)	(Axis 1, 2 ... n; see chapter 7.2.19)
	(p1, p2)	(Encoder 1, encoder 2; see chapter 7.2.20)
	(c1, c2)	(Serial interface 1, 2; see chapter 7.2.22)
	(l1)	(Linear interpolator; see chapter 7.2.21)

Programming example:

```
ld             err_all
geterror_sr    mathe
```

Function result:

If a mathematical error occurred, the function returns 1 (TRUE), otherwise it returns 0 (FALSE).

7.2.24 gethardware, Get hardware signals

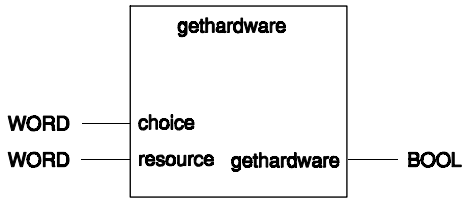
Description:

The manufacturer-defined function “gethardware” can be used for retrieving the hardware settings set with “sethardware”.

Block header:

```
Name:          gethardware
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
VAR_END

VAR_OUTPUT
  gethardware  BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice: (for axis)	pulson	Determine pulse output to power controller
	ampon	Determine power controller enable
	dirinvert	Determine sense of rotation
choice: (for serial interface)	txd_on	Determine RS 485 interface driver status
resource:	x1, x2 ... xn	Axis 1, 2 ... n
	(c1, c2)	(Serial interface 1, 2)

Programming example:

```
ld           dirinvert
gethardware   x1
```

Function result:

If the requested status is active, the function returns 1. If the status is inactive, or if an error occurred during execution, the function returns 0.

The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19 or chapter 7.2.22.

7.2.25 getmode, Get axis or encoder operating mode

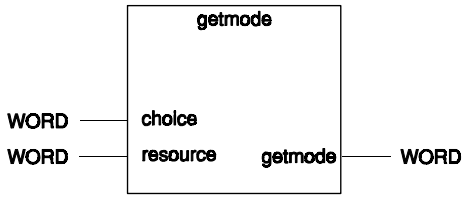
Description:

The manufacturer-defined function “getmode” is used for retrieving the operating mode of an axis or an encoder.

Block header:

```
Name:          getmode
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
VAR_END

VAR_OUTPUT
  getmode      WORD
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	drive	Indexer
	encoder	Encoder
resource:	x1, x2 ... xn	Axis 1, 2 ... n
	p1, p2	Encoder 1, encoder 2

Programming example:

```
ld          drive
getmode     x1
eq          ptp
jmpc        ptp_active
```

Comment:

Determine whether PTP_Mode is active.

Function result:

If executed successfully, the function returns the operating mode (see “setmode”) of the corresponding resource; if the function fails, it returns 0 (FALSE).
For an evaluation example, see “getensig”.

The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19 or chapter 7.2.20.

7.2.26 getnorm, Get normalizing factors (axis)

Description:

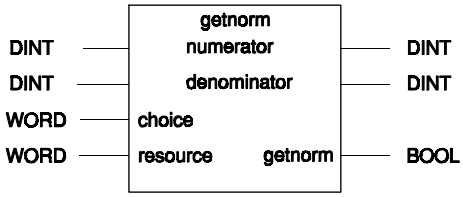
The manufacturer-defined function “getnorm” can be used for retrieving normalizing factors or gear ratios. The normalizing numerator is stored in the "numerator" parameter, the normalizing denominator in the "denominator" parameter.

NOTE
The "numerator" and "denominator" parameters are accessed via local variables (e.g. numerator, denominator).

Block header:

```
Name:           getnorm
Type:           CONTROLLER_FUN

VAR_INPUT
  choice        WORD
  resource      WORD
VAR_END

VAR_OUTPUT
  getnorm       BOOL
VAR_END

VAR_IN_OUT
  numerator     DINT
  denominator   DINT
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Erklärung:***
choice:	position	Position normalizing
	velocity	Speed normalizing
	accel	Acceleration normalizing
	source	Reference variable normalizing (electronic gear)
resource:	x1, x2 ... xn	Axis 1, 2 ... n
	(p1, p2)	(Encoder 1, encoder 2; see chapter 7.2.27)
numerator:	value	Numerator of normalizing factor
denominator:	value	Denominator of normalizing factor

Programming example:

```
VAR
  numerator         DINT
  denominator       DINT
VAR_END

ld                  position
getnorm             x1, numerator, denominator
```

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.27 getnorm, Get normalizing factors (encoder)

Description:

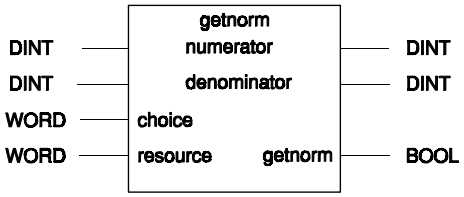
The manufacturer-defined function “getnorm” can be used for retrieving normalizing factors or gear ratios. The normalizing numerator is stored in the "numerator" parameter, the normalizing denominator in the "denominator" parameter.

NOTE
The "numerator" and "denominator" parameters are accessed via local variables (e.g. numerator, denominator).

Block header:

```
Name:           getnorm
Type:           CONTROLLER_FUN

VAR_INPUT
  choice        WORD
  resource      WORD
VAR_END

VAR_OUTPUT
  getnorm       BOOL
VAR_END

VAR_IN_OUT
  numerator     DINT
  denominator   DINT
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	position indexer	Drive units to user-defined units ratio Encoder units to drive units ratio
resource:	p1, p2 (x1, x2 ... xn)	Encoder 1, encoder 2 (Axis 1, 2 ... n; see chapter 7.2.26)
numerator:	value	Numerator of normalizing factor
denominator:	value	Denominator of normalizing factor

Programming example:

```
VAR
  numerator         DINT
  denominator       DINT
VAR_END

ld                  indexer
getnorm             p1, numerator, denominator
```

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.20.

7.2.28 getparam, Get control parameter values

Description:

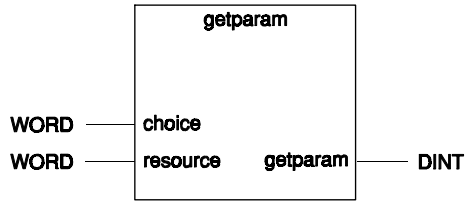
The “getparam” function can be used to retrieve the control parameters and the manipulated variable of the PID position controller.

NOTE
The “getparam” function is only applicable to series 300 units with PID position controller (e.g. WDP3-337, WDP3-338).

Block header:

```
Name:          getparam
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
VAR_END

VAR_OUTPUT
  getparam     DINT
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	kp, ki, kd, kf, ka, offset, ta command	For control parameters, see “setparam” function, Manipulated variable of position controller.
resource:	x1	Axis 1

Programming example:		***Comment:***
ld	kp	Determine the proportional component kp
getparam	x1	of axis x1

Function result:

If executed successfully, the function returns the value of the corresponding parameter or variable; if the function fails, it returns 0 (FALSE).

The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.29 getpos, Get axis positions in user-defined units

Description:

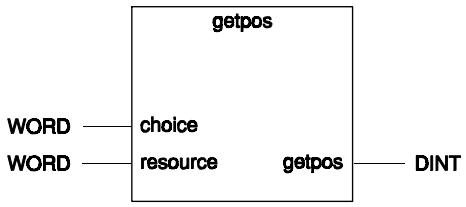
The manufacturer-defined function “getpos” can be used for retrieving axis positions in user-defined units. The user-defined positions are calculated from the drive position values using position normalizing factors. The "choice" parameter retrieves the selected position value.

NOTE
Position values for the hardware limit switches are only recognized when they are actually approached. If an undefined limit switch is referenced, an error is generated and the function returns 0.

Block header:

```
Name:          getpos
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
VAR_END

VAR_OUTPUT
  getpos       DINT
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	actual	Actual motor position
	swlimp	Value of positive software limit switch
	swlimn	Value of negative software limit switch
	target	Current setpoint
	continue	Continue position
	limp	Value of positive hardware limit switch
	limn	Value of negative hardware limit switch
	scroffset	Reference variable offset
	trig	Actual position when trigger is activated
	refsecure	Safety distance for reference movement
	srcdiff	Indexer following error
resource:	x1, x2 ... xn	Axis 1, 2 ... n
	(p1, p2)	(Encoder 1, encoder 2; see chapter 7.2.30)

Programming example:

```
ld             actual
getpos         x1
```

Function result:

If executed successfully, the function returns the corresponding position value; if the function fails, it returns 0.

The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.30 getpos, Get encoder positions in user-defined units

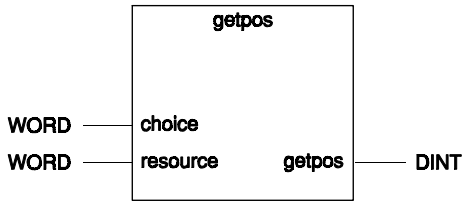
Description:

The manufacturer-defined function “getpos” can be used for retrieving encoder positions in user-defined units. The user-defined positions are calculated from the encoder position values using position normalizing factors. The "choice" parameter retrieves the selected position value.

Block header:

```
Name:          getpos
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
VAR_END

VAR_OUTPUT
  getpos       DINT
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	actual	Actual encoder position
	maxveldrag	Contouring error limit value at maximum speed
	minveldrag	Contouring error limit value at standstill
	difference	Current following error = Setpoint - Actual position
	maxdragerr	Maximum contouring error detected (can be reset using “setpos”)
	absolut [1)]	Absolute position of resolver, relative to one revolution. To interrogate the resolver position, resource p1 must be selected.

NOTE
A following error may only be retrieved if the axis resource and the encoder resource were previously assigned to each other using the manufacturer-defined function “setdrive”; otherwise a condition error will be generated.

resource:	p1, p2	Encoder 1, encoder 2
	(x1, x2 ... xn)	(Axis 1, 2 ... n; see chapter 7.2.29)

Programming example: ,

```
ld          maxveldrag
getpos      p1

ld          absolut
getpos      p1
```

Function result:

If executed successfully, the function returns the corresponding position value; if the function fails, it returns 0.

The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.20.

1) Only available for units with resolver connection: e.g. WDP3-337 and WDP3-338

7.2.31 getpos, Get axis positions in drive units

Description:

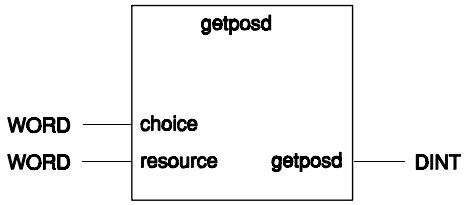
The manufacturer-defined function “getposd” can be used for retrieving axis positions in drive units. No normalization is performed. The "choice" parameter retrieves the selected position value.

NOTE
Position values for the hardware limit switches are only recognized when they are actually approached. If an undefined limit switch is referenced, an error is generated and the function returns 0.

Block header:

```
Name:          getposd
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
VAR_END

VAR_OUTPUT
  getposd      DINT
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	actual	Actual position
	swlimp	Value of positive software limit switch
	swlimn	Value of negative software limit switch
	target	Current setpoint
	continue	Continue position
	limp	Value of positive hardware limit switch
	limn	Value of negative hardware limit switch
	srcoffset	Reference variable offset
	trig	Actual position when trigger is activated
	refsecure	Safety distance for reference movement
	srcdiff	Indexer following error
resource:	x1, x2 ... xn	Axis 1, 2 ... n
	(p1, p2)	(Encoder 1, encoder 2; see chapter 7.2.32)

Programming example:

```
ld          swlimp
getposd     x1
```

Function result:

If executed successfully, the function returns the corresponding position value; if the function fails, it returns 0.

The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.32 getposd, Get encoder positions in encoder units

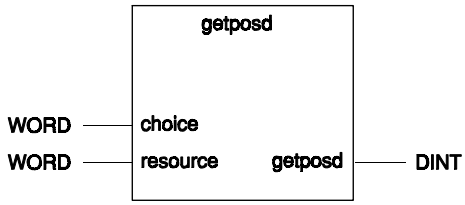
Description:

The manufacturer-defined function “getpos” can be used for retrieving encoder positions directly in encoder units.
There is no conversion using normalizing factors.

Block header:

```
Name:          getposd
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
VAR_END

VAR_OUTPUT
  getposd      DINT
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	actual	Actual encoder position
	maxveldrag	Contouring error limit value at maximum speed
	minveldrag	Contouring error limit value at standstill
	difference	Current following error = Setpoint - Actual position
	maxdragerr	Maximum contouring error detected (can be reset using “setpos”)
	absolut [1)]	Absolute position of resolver, relative to one revolution. To interrogate the resolver position, resource p1 must be selected.
resource:	p1, p2	Encoder 1, encoder 2
	(x1, x2 ... xn)	(Axis 1, 2 ... n; see chapter 7.2.31)

NOTE
A following error may only be retrieved if the axis resource and the encoder resource were previously assigned to each other using the manufacturer-defined function “setdrive”; otherwise a condition error will be generated.

Programming example: ,

```
ld          actual
getposd     p1

ld          absolut
getposd     p1
```

Function result:

If executed successfully, the function returns the encoder values directly as they are collected by the hardware using encoder units; if the function fails, it returns 0.
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.20.

1) Only available for units with resolver connection: e.g. WDP3-337 and WDP3-338

7.2.33 getsig, Get current axis signals

Description:

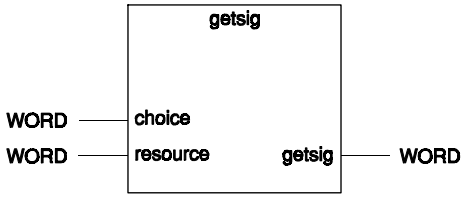
The manufacturer-defined function “getsig” retrieves the current axis signals or encoder signals (see chapter 7.2.34). The signals are taken directly from the inputs. When an input is energized, it is identified by a "1" at the corresponding bit position; when it is deenergized, by a "0".

NOTE
Since the input status cannot be determined for “swlimp”, “swlimn” and “swstop” (software switches), 0 is returned in place of the signal value.

Block header:

```
Name:          getsig
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
VAR_END

VAR_OUTPUT
  getsig_sr    WORD
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	watch	Monitoring signals
resource:	x1, x2 ... xn (p1, p2)	Axis 1, 2 ... n (Encoder 1, encoder 2; see chapter 7.2.34)

Programming example:

```
ld          watch
getsig      x1
```

Function result:

If an error occurs during execution, the function returns 0 (FALSE). If executed successfully, the function returns a bit pattern which can be evaluated using the following system constants (for an evaluation example, see “getensig”).

limp	Positive hardware limit switch	(0=deenergized)
limn	Negative hardware limit switch	(0=deenergized)
ref	Reference switch	(0=deenergized)
stop	Hardware stop input	(0=deenergized)
trig	Hardware trigger	(0=deenergized)
swlimp	Positive software limit switch	(always 0)
swlimn	Negative software limit switch	(always 0)
swstop	Software stop	(always 0)
dragerr	Contouring error	(0=contouring error)
ampready	Power controller readiness	(0=not ready)
amptemp	Power controller overtemperature	(0=overtemperature)
motortemp	Motor overtemperature	(0=overtemperature)

If the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.34 getsig, Get current encoder signals

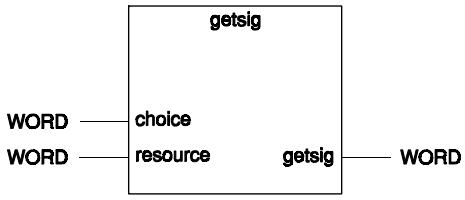
Description:

The manufacturer-defined function “getsig” retrieves the current encoder signals. The manufacturer-defined function “getsig”, as opposed to “getsig_sr”, does not have storage capability, i.e. a signal which appeared and disappeared is not shown by “getsig”.

Block header:

```
Name:          getsig
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
VAR_END

VAR_OUTPUT
  getsig_sr    WORD
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	watch	Monitoring signals
resource:	p1, p2 (x1, x2 ... xn)	Encoder 1, encoder 2 (Axis 1, 2 ... n; see chapter 7.2.33)

Programming example:

```
ld          watch
getsig      p1
```

Function result:

If an error occurs during execution, the function returns 0 (FALSE). If executed successfully, the function returns a bit pattern which can be evaluated using the following system constants (for an evaluation example, see “getensig”).

dragerr Contouring error (0=Contouring error)

The listed signals are always active high and cannot be modified.

dragerr is set whenever the rotation monitoring feature is active and a contouring error is detected.

The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.20.

7.2.35 getsig_activ_h, Get active state of axis signals

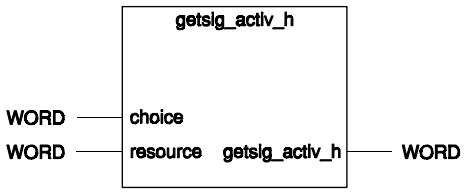
Description:

The manufacturer-defined function “getsig_activ_h” reads the active states of the predefined signal inputs. If the bit position of the corresponding signal is 0, it is interpreted to be active when the input is deenergized. If the bit position is 1, the signal is interpreted to be active when the input is energized. The active states of the signal inputs can be set using the manufacturer-defined function “setsig_activ_h”.

Block header:

```
Name:            getsig_activ_h
Type:            CONTROLLER_FUN

VAR_INPUT
  choice         WORD
  resource       WORD
VAR_END

VAR_OUTPUT
  getsig_activ_h WORD
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	watch	Monitoring signals
resource:	x1, x2 ... xn	Axis 1, 2 ... n

Programming example:

```
ld               watch
getsig_activ_h   x1
```

Function result:

If an error occurs during execution, the function returns 0 (FALSE). If executed successfully, the function returns a bit pattern which can be evaluated using the following system constants (for an evaluation example, see “getensig”).

limp	Positive hardware limit switch
limn	Negative hardware limit switch
ref	Reference switch
trig	Hardware trigger

The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.36 getsig_sr, Get temporarily stored axis signals

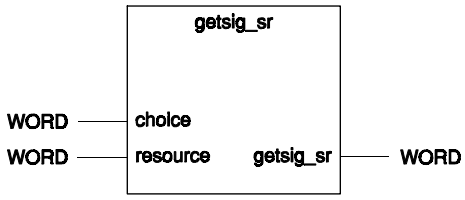
Description:

The manufacturer-defined function “getsig_sr” can be used to retrieve temporarily stored axis signals (signal flags). The signal states are stored in the flag bits. A signal is stored until it is reset using the manufacturer-defined function “clrsig_sr”. A flag bit is set to "1" when the corresponding signal is active. The active state of a signal can be set with the manufacturer-defined function “setsig_activ_h”.

Block header:

```
Name:          getsig_sr
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
VAR_END

VAR_OUTPUT
  getsig_sr    WORD
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	watch	Monitoring signals
resource:	x1, x2 ... xn (p1, p2)	Axis 1, 2 ... n (Encoder 1, encoder 2; see chapter 7.2.37)

Programming example:

```
ld          watch
getsig_sr   x1
and         limp
eq          limp
jmpc        error
```

Comment:

If positive limit switch is active, jump to “error” label.

Function result:

If an error occurs during execution, the function returns 0 (FALSE). If executed successfully, the function returns a bit pattern which can be evaluated using the following system constants (for an evaluation example, see “getensig”).

limp	Positive hardware limit switch
limn	Negative hardware limit switch
ref	Reference switch
stop	Hardware stop input
trig	Hardware trigger
swlimp	Positive software limit switch
swlimn	Negative software limit switch
swstop	Software stop
dragerr	Contouring error
encerr	Encoder error
ampnotready	Power controller not ready
amptemp	Power controller overtemperature
motortemp	Motor overtemperature

The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.37 getsig_sr, Get temporarily stored encoder signals

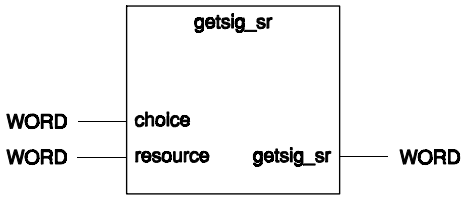
Description:

The manufacturer-defined function “getsig_sr” can be used to retrieve temporarily stored encoder signals (signal flags). The signal states are stored in the flag bits. A signal is stored until it is reset using the manufacturer-defined function “clrsig_sr”. A flag bit is set to "1" when the corresponding signal is active.

Block header:

```
Name:          getsig_sr
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
VAR_END

VAR_OUTPUT
  getsig_sr    WORD
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	watch	Monitoring signals
resource:	p1, p2 (x1, x2 ... xn)	Encoder 1, encoder 2 (Axis 1, 2 ... n; see chapter 7.2.36)

Programming example:

```
ld          watch
getsig_sr   p1
and         dragerr
eq          dragerr
jmpc        error
```

Comment:

If a contouring error is detected, jump to “error” label.

Function result:

If an error occurs during execution, the function returns 0 (FALSE). If executed successfully, the function returns a bit pattern which can be evaluated using the following system constants:

dragerr	Contouring error	(1=Contouring error)
encerr	Encoder error	(1=Encoder error)
index [1)]	Index pulse	(1=Index pulse)

The listed signals are always active high and cannot be modified.

dragerr is set whenever the rotation monitoring feature is active and a contouring error is detected.
encerr is set when a hardware error is detected (e.g. encoder connector unplugged).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.20.

1) Only available for resource p2

7.2.38 getstate, Get axis status

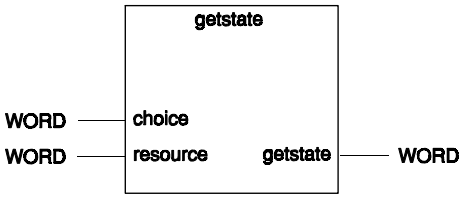
Description:

The manufacturer-defined function “getstate” retrieves axis status conditions. The "choice" parameter can be used to select different status words.

Block header:

```
Name:          getstate
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
VAR_END

VAR_OUTPUT
  getstate     WORD
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	action	Drive action states
	motion	Movement states
	warning	Warnings
	ref	Reference movement states
resource:	x1, x2 ... xn	Axis 1, 2 ... n
	(c1, c2)	(Serial interface 1, 2; see chapter 7.2.40)
	(batt)	(Battery; see chapter 7.2.41)
	(keys)	(Front panel keys; see chapter 7.2.42)
	(l1)	(Linear interpolator; see chapter 7.2.39)

Programming example:		***Comment:***
ld	action	If the drive status is "breaked",
getstate	x1	jump to “error” label.
and	breaked	
jmpc	error	

Function result:

If executed successfully, the function stores a WORD type result in the CR, depending on the resource and choice parameters. If the function fails, 0 (FALSE) is stored in the CR. The individual bits have the following meanings:

choice:	***Value/constant:***	***Explanation:***
action	notactive	Axis is inactive or at a standstill, awaiting commands
	active	Axis is moving, or active
	breaked	Axis is breaked (interrupted, e.g. by stop or limit switch)
	blocked	Axis is blocked
motion	accel	Axis accelerates
	constant	Axis moves at constant speed
	brake	Axis decelerates
	stand	Axis is at a standstill
	errbrake	Axis decelerates at emergency deceleration ramp
	plus	Axis moves in positive direction
	minus	Axis moves in negative direction
warning	ampdis	Power drive not enabled
	posundef	Drive actual position not defined
	accundef	Acceleration not defined
	posover	Position overrun occurred
ref	refok	Reference movement successfully completed
	referr	Error during reference movement
	refactiv	Reference movement in progress
	refblocked	Reference movement error due to Axis blocked
	refnoenable	Reference movement limit switch not enabled
	limp	Reference movement error due to positive limit switch
	limn	Reference movement error due to negative limit switch
	ref	Reference movement error due to additional limit switch
	stop	Reference movement error due to hardware stop

The "geterror_sr" function can be used to retrieve any error condition, see chapter 7.2.19.

The action status conditions (choice: action) of the drive are defined as follows:

– not active

  In point-to-point or speed mode,, the axis is “notactive” when the shaft is at a standstill.
  In position following mode,, the axis is “notactive” when the shaft is at a standstill and has any of the action status conditions “breaked” or “blocked” (see below).

– active

  The axis is “active” when a movement is carried out. In position following mode, the axis is also “active” when the shaft does not move, because the reference variable does not change. The axis assumes “notactive” status only when the shaft is “breaked” or “blocked”. The "choice: motion" parameter can be used for determining the movement status of the axis.

– breaked

  The axis has “breaked” status if any of the following events caused an interruption of the movement:

  – Software limit switch position or hardware limit switch reached
  – Reference switch reached
  – Software (stop command) or hardware stop triggered

– blocked

  The axis has “blocked” status if any of the following events caused an interruption of the movement:

  – Contouring error detected
  – Encoder error detected
  – Power controller not ready
  – Power controller or motor overtemperature

An axis with status “breaked” or “blocked” can be reenabled by calling the manufacturer-defined function “clrsig_sr”.

7.2.39 getstate, Get linear interpolation status conditions

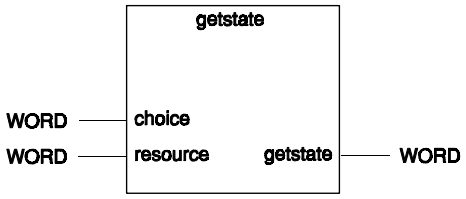
Description:

The manufacturer-defined function “getstate” retrieves status conditions in a linear interpolation process. The "choice" parameter can be used to select different status words.

Block header:

```
Name:          getstate
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
VAR_END

VAR_OUTPUT
  getstate     WORD
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	action	Drive action states
	motion	Movement states
resource:	l1	Linear interpolator
	(x1, x2 ... xn)	(Axis 1, 2 ... n; see chapter 7.2.38)
	(c1, c2)	(Serial interface 1, 2; see chapter 7.2.40)
	(batt)	(Battery; see chapter 7.2.41)
	(keys)	(Front panel keys; see chapter 7.2.42)

Programming example:

```
ld          action
getstate    x1
and         breaked
jmpc        error
```

Comment:

If the drive status is "breaked", jump to “error” label.

Function result:

If executed successfully, the function stores a WORD type result in the CR, depending on the "resource" and "choice" parameters. If the function fails, 0 (FALSE) is stored in the CR. The individual bits have the following meanings:

choice:	***Wert / Konstante:***	***Erklärung:***
action	notactive	Interpolation is inactiv
	active	Interpolation ist aktiv
	breaked	One or more axis interrupted (e.g. by stop or limit switch)
	blocked	One or more axis axes blocked.
motion	accel	Reference axis [1] accelerates
	constant	Reference axis [1] moves at constant speed
	brake	Reference axis [1] decelerates
	stand	Reference axis [1] at standstill or linear interpolation finished
	errbrake	Reference axis [1] decelerates at emergency braking ramp
	plus	Reference axis [1] moves in positive direction
	minus	Reference axis [1] moves in negative direction

The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

1) The reference axis is the one which travels the longer distance in a linear interpolation process.

The action status conditions (choice: action) of the axes are defined as follows:

– breaked

  An axis has “breaked” status if any of the following events caused an interruption of the movement:

  – Software limit switch position or hardware limit switch reached
  – Reference switch reached
  – Software (stop command) or hardware stop triggered

– blocked

  An axis has “blocked” status if any of the following events caused an interruption of the movement:

  – Contouring error detected
  – Encoder error detected
  – Power controller not ready
  – Power controller or motor overtemperature

An axis with status “breaked” or “blocked” can be reenabled by calling the manufacturer-defined function “clrsig_sr”.

NOTE
If an axis is “breaked” or “blocked”,, the “notactive” status is always set (linear interpola tion inactive).

7.2.40 getstate, Get serial interface status

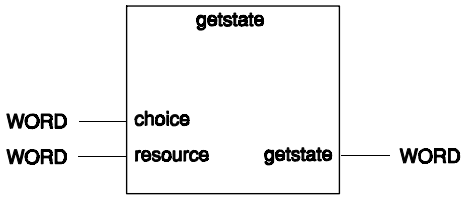
Description:

A special selection parameter is available for retrieving the status of the serial interface 1 or 2 resource. When calling com_init_asc, receive_char, receive_string, send_char, send_string the same status flags are returned in the return status of the respective function except those flags marked [1] below.

Block header:

```
Name:          getstate
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
VAR_END

VAR_OUTPUT
  getstate     WORD
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	com	Interface status
resource:	c1, c2 (x1, x2 ... xn) (batt) (keys)	Serial interface 1, 2 (Axis 1, 2 ... n; see chapter 7.2.38) (Battery; see chapter 7.2.41) (Front panel keys; see chapter 7.2.42)

Programming example:

```
ld          com
getstate    c2
```

Function result:

If an error occurs during execution, the function returns 0 (FALSE). If executed successfully, the function returns a bit pattern which can be evaluated using the following system constants (for an evaluation example, see “getensig”).

choice:	***Value/constant:***	***Explanation:***
com	endcharnotfound	String end flag not found
	recfifoempty	Receive buffer empty
	sendfifofull	Transmit buffer full
	sfifoemptyakt	Transmit buffer empty [1]
	rfifoemptyakt	Receive buffer empty [1]
	erroraktiv	Error occurred (retrieve with geterror_sr, clear with clrerror_sr) [2]

The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.22.

1) The current status at retrieval time is output instead of the value determined at the last call of any of the following manufacturer-defined functions: com_init_asc, receive_char, receive_string, send_char, send_string. For example, when the manufacturer-defined function receive_char is called, the recfifoempty status is set. If a character is received subsequently, the manufacturer-defined function getstate reports this by rfifoemptyakt = FALSE. (recfifoempty = TRUE corresponds to the status of the previous receive_char call).

2) This flag indicates that an error occurred while processing the ASC component. You can use geterror_sr with the appropriate selection parameter to read the error word and you can clear it using clrerror_sr.

7.2.41 getstate, Get battery status

Description:

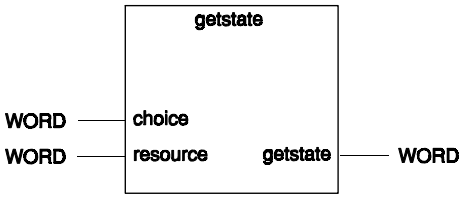
The manufacturer-defined function “getstate” retrieves status conditions. The "choice" parameter can be used to select different status words.

Block header:

Name: getstate
Type: CONTROLLER_FUN

```
VAR_INPUT
  choice      WORD
  resource    WORD
VAR_END

VAR_OUTPUT
  getstate    WORD
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	none	No options available
resource:	batt (x1, x2 ... xn) (c1, c2) (keys)	Battery (Axis 1, 2 ... n; see chapter 7.2.38) (Serial interface 1, 2; see chapter 7.2.40) (Front panel keys; see chapter 7.2.42)

Programming example:

```
ld          none
getstate    batt
```

Function result:

If executed successfully, the function returns 1 (battery o.k.) or 0 (battery voltage low).

7.2.42 getstate, Get front panel key status

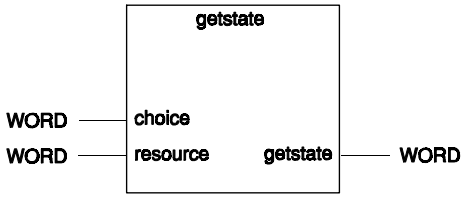
Description:

The manufacturer-defined function “getstate” retrieves front panel key status conditions. The "choice" parameter can be used to select different status words. If the "keyfree" parameter is specified, the keys are enabled for normal use. If the "keyall" parameter is specified, the normal functionality of the keys is disabled, and the keys are only accessible from the application program.

Block header:

```
Name:          getstate
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
VAR_END

VAR_OUTPUT
  getstate     WORD
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	keyfree	Front panel keys enabled for normal use (see controller manual)
	keyall	Check all front panel keys and enable them for application program
resource:	keys	Front panel keys
	(x1, x2 ... xn)	(Axis 1, 2 ... n; see chapter 7.2.38)
	(c1, c2)	(Serial interface 1, 2; see chapter 7.2.40)
	(batt)	(Battery; see chapter 7.2.41)

Programming example:

```
ld          keyall
getstate    keys
```

Function result:

If executed successfully with the "keyfree" parameter, the function always returns “no key pressed”.

If executed successfully with the "keyall" parameter, the function returns a bit list indicating the status of the keys. The following constants can be used for evaluation:

keyul	Keys on upper left
keyur	Keys on upper right
keyll	Keys on lower left
keylr	Keys on lower right

7.2.43 getvel, Get speed values

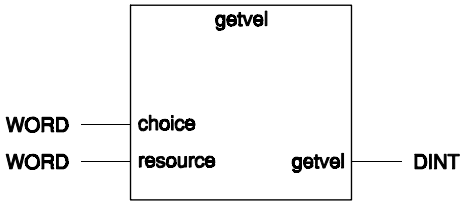
Description:

The manufacturer-defined function “getvel” retrieves the speed values. The "choice" parameter determines the speed value to be retrieved.

Block header:

```
Name:          getvel
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
VAR_END

VAR_OUTPUT
  getvel       DINT
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	actual	Current speed
	target	Set speed
	start	Start speed
	system	Maximum system speed
	continue	Set speed before movement interruption
resource:	x1, x2 ... xn	Axis 1, 2 ... n

Programming example:

```
ld          actual
getvel      x1
```

Function result:

If executed successfully, the function returns the corresponding speed value; if the function fails, it returns 0 (FALSE).

The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.44 linmove2, Relative linear interpolation with two axes

Description:

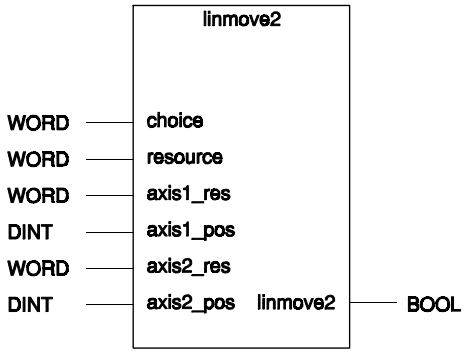
The “linmove2” function can be used to perform linear interpolation with two axes in a system of relative dimensions from a point A to a point B. The position values are specified in user-defined units.

NOTE
The following presettings are required for interpolation:
- *The axes involved in interpolation must be set to point-to-point mode (see “setmode” function in controller library).*
- *The speeds and accelerations of the axes involved must be set before the interpolation process (see “vel” and “accel” functions in controller library).*

Block header:

```
Name:          linmove2
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
  axis1_res    WORD
  axis1_pos    DINT
  axis2_res    WORD
  axis2_pos    DINT
VAR_END

VAR_OUTPUT
  linmove2     BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	0	Not relevant in the current release
resource:	l1	Linear interpolator
axis1_res:	x1, x2 ... xn	First axis: Axis 1, 2 ... n
axis1_pos:	Position value	Target position of first axis
axis2_res:	x1, x2 ... xn	Second axis: Axis 1, 2 ... n
axis2_pos:	Position value	Target position of second axis

Programming example:		***Comment:***
ld	0	Position axes 1 and 3 with linear
linmove2	l1, x1, 1000, x3, 5000	interpolation by 1000/5000 user-defined units

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.21.

7.2.45 linmove2w, Relative linear interpolation with two axes and wait for end of movement

Description:

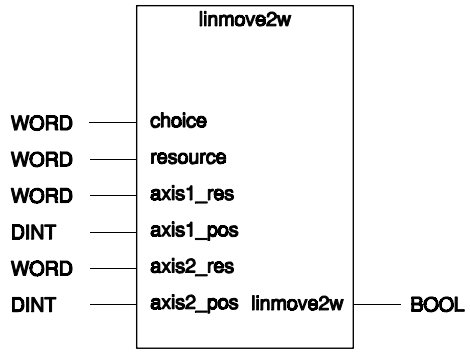
The “linmove2w” function can be used to perform linear interpolation with two axes in a system of relative dimensions from a point A to a point B. The program is stopped until the movement has been completed. The position values are specified in user-defined units.

NOTE
The following presettings are required for interpolation:
- *The axes involved in interpolation must be set to point-to-point mode (see “setmode” function in controller library).*
- *The speeds and accelerations of the axes involved must be set before the interpolation process (see “vel” and “accel” functions in controller library).*

Block header:

```
Name:          linmove2w
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
  axis1_res    WORD
  axis1_pos    DINT
  axis2_res    WORD
  axis2_pos    DINT
VAR_END

VAR_OUTPUT
  linmove2w    BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	0	Not relevant in the current release
resource:	l1	Linear interpolator
axis1_res:	x1, x2 ... xn	First axis: Axis 1, 2 ... n
axis1_pos:	Position value	Target position of first axis
axis2_res:	x1, x2 ... xn	Second axis: Axis 1, 2 ... n
axis2_pos:	Position value	Target position of second axis

Programming example:	***Comment:***
ld 0 linmove2w l1, x1, 1000, x3, 5000	Position axes 1 and 3 with linear interpolation by 1000/5000 user-defined units and wait for end of movement

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.21.

7.2.46 linmove3, Relative linear interpolation with three axes

Description:

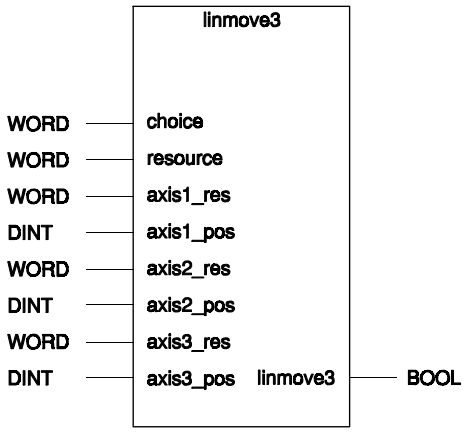
The “linmove3” function can be used to perform linear interpolation with three axes in a system of relative dimensions from a point A to a point B. The position values are specified in user-defined units.

NOTE
The following presettings are required for interpolation:
- *The axes involved in interpolation must be set to point-to-point mode (see “setmode” function in controller library).*
- *The speeds and accelerations of the axes involved must be set before the interpolation process (see “vel” and “accel” functions in controller library).*

Block header:

```
Name:          linmove3
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
  axis1_res    WORD
  axis1_pos    DINT
  axis2_res    WORD
  axis2_pos    DINT
  axis3_res    WORD
  axis3_pos    DINT
VAR_END

VAR_OUTPUT
  linmove3     BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	0	Not relevant in the current release
resource:	l1	Linear interpolator
axis1_res:	x1, x2 ... xn	First axis: Axis 1, 2 ... n
axis1_pos:	Position value	Target position of first axis
axis2_res:	x1, x2 ... xn	Second axis: Axis 1, 2 ... n
axis2_pos:	Position value	Target position of second axis
axis3_res:	x1, x2 ... xn	Third axis: Axis 1, 2 ... n
axis3_pos:	Position value	Target position of third axis

Programming example:		***Comment:***
ld	0	Position axes 1, 2 and 4 with linear
linmove3	l1, x1, 1000, x2, 5000, x4, 2000	interpolation by 1000/5000/2000 user-defined units

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.21.

7.2.47 linmove3w, Relative linear interpolation with three axes and wait for end of movement

Description:

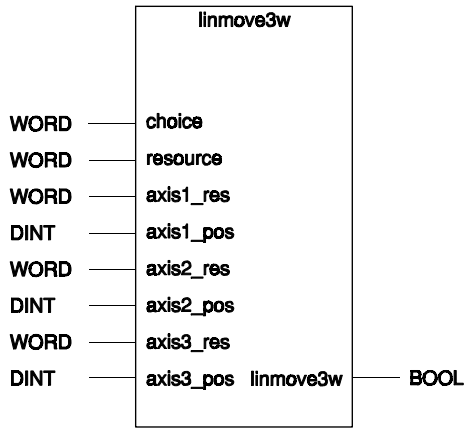
The “linmove3w” function can be used to perform linear interpolation with three axes in a system of relative dimensions from a point A to a point B. The program is stopped until the movement has been completed. The position values are specified in user-defined units.

NOTE
The following presettings are required for interpolation:
- *The axes involved in interpolation must be set to point-to-point mode (see “setmode” function in controller library).*
- *The speeds and accelerations of the axes involved must be set before the interpolation process (see “vel” and “accel” functions in controller library).*

Block header:

```
Name:          linmove3w
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
  axis1_res    WORD
  axis1_pos    DINT
  axis2_res    WORD
  axis2_pos    DINT
  axis3_res    WORD
  axis3_pos    DINT
VAR_END

VAR_OUTPUT
  linmove3w    BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	0	Not relevant in the current release
resource:	I1	Linear interpolator
axis1_res:	x1, x2 ... xn	First axis: Axis 1, 2 ... n
axis1_pos:	Position value	Target position of first axis
axis2_res:	x1, x2 ... xn	Second axis: Axis 1, 2 ... n
axis2_pos:	Position value	Target position of second axis
axis3_res:	x1, x2 ... xn	Third axis: Axis 1, 2 ... n
axis3_pos:	Position value	Target position of third axis

Programming example:		***Comment:***
ld	0	Position axes 1, 2 and 4 with linear interpolation by 1000/5000/2000 user-defined units and wait for end of movement
linmove3w	I1, x1, 1000, x2, 5000, x4, 2000	

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.21.

7.2.48 linpos2, Absolute linear interpolation with two axes

Description:

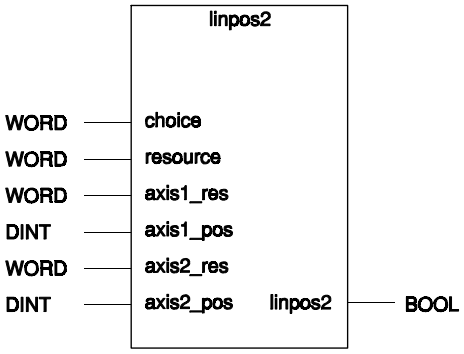
The “linpos2” function can be used to perform linear interpolation with two axes in a system of absolute dimensions from a point A to a point B. The position values are specified in user-defined units.

NOTE
The following presettings are required for interpolation:
- *The axes involved in interpolation must be set to point-to-point mode (see “setmode” function in controller library).*
- *The speeds and accelerations of the axes involved must be set before the interpolation process (see “vel” and “accel” functions in controller library).*

Block header:

```
Name:          linpos2
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
  axis1_res    WORD
  axis1_pos    DINT
  axis2_res    WORD
  axis2_pos    DINT
VAR_END

VAR_OUTPUT
  linpos2      BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	0	Not relevant in the current release
resource:	l1	Linear interpolator
axis1_res:	x1, x2 ... xn	First axis: Axis 1, 2 ... n
axis1_pos:	Position value	Target position of first axis
axis2_res:	x1, x2 ... xn	Second axis: Axis 1, 2 ... n
axis2_pos:	Position value	Target position of second axis

Programming example:		***Comment:***
ld	0	Position axes 1 and 3 with linear interpolation to point 1000/5000 in absolute system of dimensions
linpos2	l1, x1, 1000, x3, 5000	

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.21.

7.2.49 linpos2w, Absolute linear interpolation with two axes and wait for end of movement

Description:

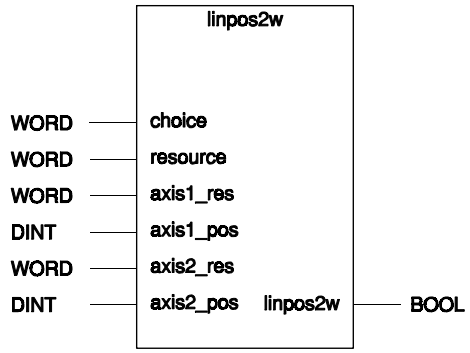
The “linpos2w” function can be used to perform linear interpolation with two axes in a system of absolute dimensions from a point A to a point B. The program is stopped until the movement has been completed. The position values are specified in user-defined units.

NOTE
The following presettings are required for interpolation:
- *The axes involved in interpolation must be set to point-to-point mode (see “setmode” function in controller library).*
- *The speeds and accelerations of the axes involved must be set before the interpolation process (see “vel” and “accel” functions in controller library).*

Block header:

```
Name:          linpos2w
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
  axis1_res    WORD
  axis1_pos    DINT
  axis2_res    WORD
  axis2_pos    DINT
VAR_END

VAR_OUTPUT
  linpos2w     BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	0	Not relevant in the current release
resource:	l1	Linear interpolator
axis1_res:	x1, x2 ... xn	First axis: Axis 1, 2 ... n
axis1_pos:	Position value	Target position of first axis
axis2_res:	x1, x2 ... xn	Second axis: Axis 1, 2 ... n
axis2_pos:	Position value	Target position of second axis

Programming example:		***Comment:***
ld	0	Position axes 1 and 3 with linear interpolation to point 1000/5000 in absolute system of dimensions and wait for end of movement
linpos2w	l1, x1, 1000, x3, 5000	

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.21.

7.2.50 linpos3, Absolute linear interpolation with three axes

Description:

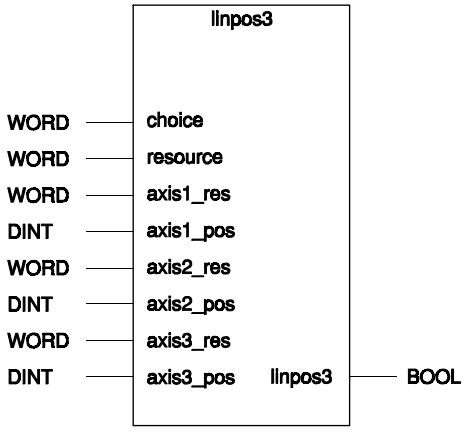
The “linpos3” function can be used to perform linear interpolation with three axes in a system of absolute dimensions from a point A to a point B. The position values are specified in user-defined units.

NOTE
The following presettings are required for interpolation:
- *The axes involved in interpolation must be set to point-to-point mode (see “setmode” function in controller library).*
- *The speeds and accelerations of the axes involved must be set before the interpolation process (see “vel” and “accel” functions in controller library).*

Block header:

```
Name:          linpos3
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
  axis1_res    WORD
  axis1_pos    DINT
  axis2_res    WORD
  axis2_pos    DINT
  axis3_res    WORD
  axis3_pos    DINT
VAR_END

VAR_OUTPUT
  linpos3      BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	0	Not relevant in the current release
resource:	l1	Linear interpolator
axis1_res:	x1, x2 ... xn	First axis: Achse 1, 2 ... n
axis1_pos:	Position value	Target position of first axis
axis2_res:	x1, x2 ... xn	Second axis: Axis 1, 2 ... n
axis2_pos:	Position value	Target position of second axis
axis3_res:	x1, x2 ... xn	Third axis: Axis 1, 2 ... n
axis3_pos:	Position value	Target position of third axis

Programming example:		***Comment:***
ld	0	Position axes 1, 2 and 4 with linear interpolation to point 1000/5000/2000 in absolute system of dimensions
linpos3	l1, x1, 1000, x2, 5000, x4, 2000	

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.21.

7.2.51 linpos3w, Absolute linear interpolation with three axes and wait for end of movement

Description:

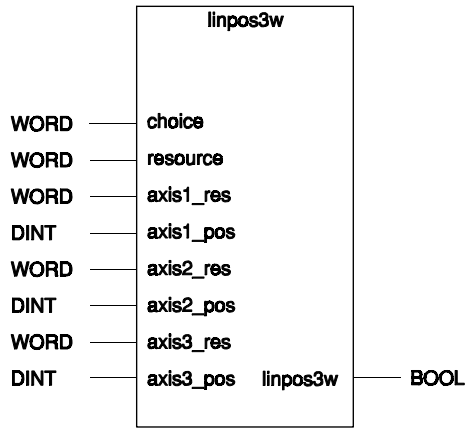
The “linpos3w” function can be used to perform linear interpolation with three axes in a system of absolute dimensions from a point A to a point B. The program is stopped until the movement has been completed. The position values are specified in user-defined units.

NOTE
The following presettings are required for interpolation:
- *The axes involved in interpolation must be set to point-to-point mode (see “setmode” function in controller library).*
- *The speeds and accelerations of the axes involved must be set before the interpolation process (see “vel” and “accel” functions in controller library).*

Block header:

```
Name:          linpos3w
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
  axis1_res    WORD
  axis1_pos    DINT
  axis2_res    WORD
  axis2_pos    DINT
  axis3_res    WORD
  axis3_pos    DINT
VAR_END

VAR_OUTPUT
  linpos3w     BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	0	Not relevant in the current release
resource:	I1	Linear interpolator
axis1_res:	x1, x2 ... xn	First axis: Axis 1, 2 ... n
axis1_pos:	Position value	Target position of first axis
axis2_res:	x1, x2 ... xn	Second axis: Axis 1, 2 ... n
axis2_pos:	Position value	Target position of second axis
axis3_res:	x1, x2 ... xn	Third axis: Axis 1, 2 ... n
axis3_pos:	Position value	Target position of third axis

Programming example:		***Comment:***
ld	0	Position axes 1, 2 and 4 with linear interpolation to point 1000/5000/2000 in absolute system of dimensions and wait for end of movement
linpos3w	I1, x1, 1000, x2, 5000, x4, 2000	

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.21.

7.2.52 list, Disable position list processing

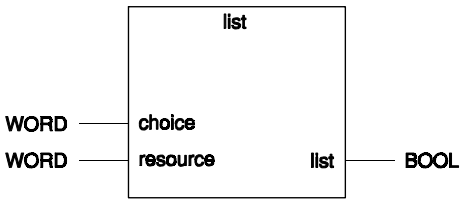
Description:

The manufacturer-defined function “list” stops processing of a position list (see chapter 3.3.5 "Positions").

Block header:

```
Name:          list
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
VAR_END

VAR_OUTPUT
  list         BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	kill	Stop position list processing
resource:	x1, x2 ... xn	Axis 1, 2 ... n

Programming example:

```
ld          kill
list        x1, x2 ... xn
```

Function result:

If executed successfully, the function returns 1 (TRUE), otherwise it returns 0 (FALSE).

7.2.53 move, Relative positioning in user-defined units

Description:

The manufacturer-defined function “move” is used in point-to-point mode for presetting a relative target position in user-defined units and starting a positioning operation. From the specified relative target position, a new absolute target position is calculated. Program execution continues in parallel, and the preset position can be changed during positioning. To calculate the new target position, the position value is added to the previous target position.

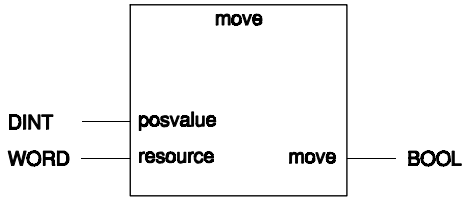
NOTE
The following requirements must be met before specifying a new target position:
- The axis must be in point-to-point mode.
- The acceleration must be defined.
- The axis must not be "blocked".
If any of these requirements is not met, an error is generated and the new target position is ignored.

NOTE
Calling the “move” function will result in a movement if the drive is not interrupted. The target position will be accepted even when the drive is interrupted.

Block header:

```
Name:          move
Type:          CONTROLLER_FUN

VAR_INPUT
  posvalue     DINT
  resource     WORD
VAR_END

VAR_OUTPUT
  move         BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
posvalue:	Position value	Relative target position in user-defined units
resource:	x1, x2 ... xn	Axis 1, 2 ... n

Programming example:

```
ld             5000
move           x1
```

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).

The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.54 moved, Relative positioning in drive units

Description:

The manufacturer-defined function “moved” is used in point-to-point mode for presetting a relative target position in drive units and starting a positioning operation. From the specified relative target position, a new absolute target position is calculated. Program execution continues in parallel, and the preset position can be changed during positioning. To calculate the new target position, the position value is added to the previous target position.

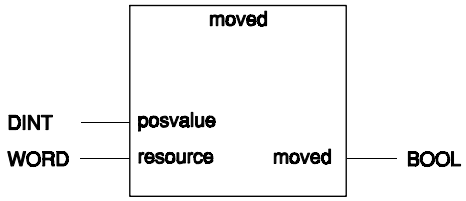
NOTE
The following requirements must be met before specifying a new target position:
- The axis must be in point-to-point mode.
- The acceleration must be defined.
- The axis must not be "blocked".
If any of these requirements is not met, an error is generated and the new target position is ignored.

NOTE
Calling the “moved” function will result in a movement if the drive is not interrupted. The target position will be accepted even when the drive is interrupted.

Block header:

```
Name:          moved
Type:          CONTROLLER_FUN

VAR_INPUT
  posvalue     DINT
  resource     WORD
VAR_END

VAR_OUTPUT
  moved        BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
posvalue:	Position value	Relative target position in drive units
resource:	x1, x2 ... xn	Axis 1, 2 ... n

Programming example:

```
ld          5000
moved       x1
```

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).

The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.55 movedf, Relative positioning in drive units and wait until position reached

Description:

The manufacturer-defined function “movedf” is used in point-to-point mode to preset a relative target position in drive units and wait until the target position has been reached. From the specified relative target position, a new absolute target position is calculated. The controlling task is stopped during the movement.

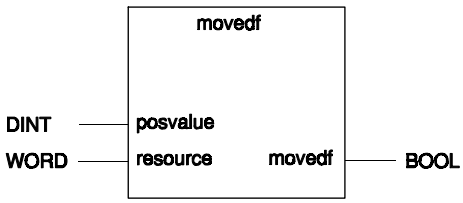
NOTE
The “movedf” function does not return a result until the setpoint has been reached.
If an interruption occurs during a movement, the function is not terminated.

NOTE
The following requirements must be met before specifying a new target position:
- The axis must be in point-to-point mode.
- The acceleration must be defined.
- The axis must not be interrupted.
- The axis must not be "blocked".
If any of these requirements is not met, an error is generated and the new target position is ignored.

Block header:

```
Name:          movedf
Type:          CONTROLLER_FUN

VAR_INPUT
  posvalue     DINT
  resource     WORD
VAR_END

VAR_OUTPUT
  movedf       BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
posvalue:	Position value	Relative target position in drive units
resource:	x1, x2 ... xn	Axis 1, 2 ... n

Programming example:

```
ld          5000
movedf      x1
```

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).

The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.56 movedw, Relative positioning in drive units and wait for end of movement

Description:

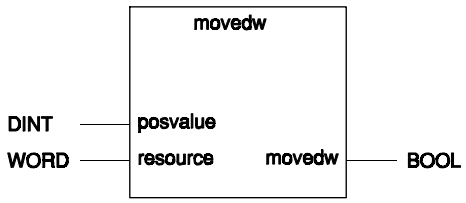
The manufacturer-defined function “movedw” is used in point-to-point mode to preset a relative target position in drive units and wait until the movement has been completed or interrupted. From the specified relative target position, a new absolute target position is calculated. The controlling task is stopped during the movement.

NOTE
The following requirements must be met before specifying a new target position:
- The axis must be in point-to-point mode.
- The acceleration must be defined.
- The axis must not be interrupted.
- The axis must not be "blocked".
If any of these requirements is not met, an error is generated and the new target position is ignored.

Block header:

```
Name:          movedw
Type:          CONTROLLER_FUN

VAR_INPUT
  posvalue     DINT
  resource     WORD
VAR_END

VAR_OUTPUT
  movedw       BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
posvalue:	Position value	Relative target position in drive units
resource:	x1, x2 ... xn	Axis 1, 2 ... n

Programming example:

```
ld          5000
movedw      x1
```

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE). If the movement is interrupted (e.g. by stop), it returns 0.

The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.57 movef, Relative positioning in user-defined units and wait until position reached

Description:

The manufacturer-defined function “movef” is used in point-to-point mode to preset a relative target position in user-defined units and wait until the target position has been reached. From the specified relative target position, a new absolute target position is calculated. The controlling task is stopped during the movement.

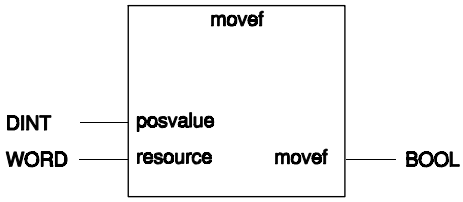
NOTE
The “movef” function does not return a result until the setpoint has been reached.
If an interruption occurs during a movement, the function is not terminated.

NOTE
The following requirements must be met before specifying a new target position:
- The axis must be in point-to-point mode.
- The acceleration must be defined.
- The axis must not be interrupted.
- The axis must not be "blocked".
If any of these requirements is not met, an error is generated and the new target position is ignored.

Block header:

```
Name:          movef
Type:          CONTROLLER_FUN

VAR_INPUT
  posvalue     DINT
  resource     WORD
VAR_END

VAR_OUTPUT
  movef        BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
posvalue:	Position value	Relative target position in user-defined units
resource:	x1, x2 ... xn	Axis 1, 2 ... n

Programming example:

```
ld          50000
movef       x1
```

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).

The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.58 movew, Relative positioning in user-defined units and wait for end of movement

Description:

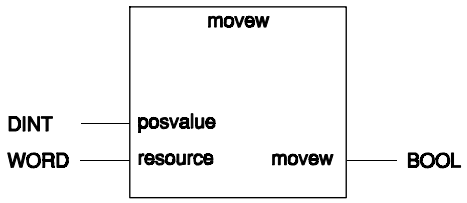
The manufacturer-defined function “movew” is used in point-to-point mode to preset a relative target position in user-defined units and wait until the movement has been completed or interrupted. From the specified relative target position, a new absolute target position is calculated. The controlling task is stopped during the movement.

NOTE
The following requirements must be met before specifying a new target position:
- The axis must be in point-to-point mode.
- The acceleration must be defined.
- The axis must not be interrupted.
- The axis must not be "blocked".
If any of these requirements is not met, an error is generated and the new target position is ignored.

Block header:

```
Name:          movew
Type:          CONTROLLER_FUN

VAR_INPUT
  posvalue     DINT
  resource     WORD
VAR_END

VAR_OUTPUT
  movew        BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
posvalue:	Position value	Relative target position in user-defined units
resource:	x1, x2 ... xn	Axis 1, 2 ... n

Programming example:

```
ld          50000
movew       x1
```

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE). If the movement is interrupted (e.g. by stop), it returns 0.

The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.59 pos, Absolute positioning in user-defined units

Description:

The manufacturer-defined function “pos” is used in point-to-point mode for presetting an absolute target position in user-defined units and starting a positioning operation. Program execution continues in parallel, and the preset position can be changed during positioning.

NOTE
The following requirements must be met before specifying a new target position:
- *The actual position must be defined.*
- *The axis must be in point-to-point mode.*
- *The acceleration must be defined.*
- *A reference movement must not be active.*
- *The axis must not be "blocked".*

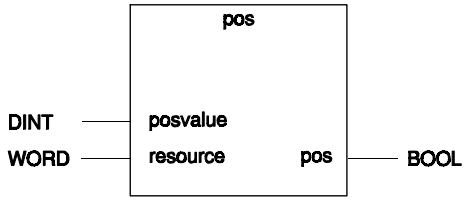
If any of these requirements is not met, an error is generated and the new target position is ignored.

NOTE
Calling the “pos” function will result in a movement if the drive is not interrupted. The target position will be accepted even when the drive is interrupted.

Block header:

```
Name:          pos
Type:          CONTROLLER_FUN

VAR_INPUT
  posvalue     DINT
  resource     WORD
VAR_END

VAR_OUTPUT
  pos          BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
posvalue:	Position value	Target position in user-defined units
resource:	x1, x2 ... xn	Axis 1, 2 ... n

Programming example:

```
ld             5000
pos            x1
```

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.60 posd, Absolute positioning in drive units

Description:

The manufacturer-defined function “posd” is used in point-to-point mode for presetting an absolute target position in drive units and starting a positioning operation. Program execution continues in parallel, and the preset position can be changed during positioning.

NOTE
The following requirements must be met before specifying a new target position:
- *The actual position must be defined.*
- *The axis must be in point-to-point mode.*
- *The acceleration must be defined.*
- *A reference movement must not be active.*
- *The axis must not be "blocked".*

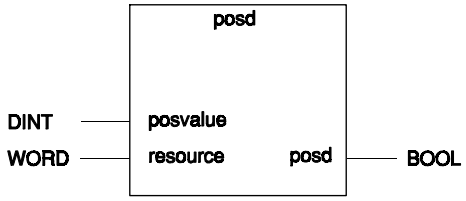
If any of these requirements is not met, an error is generated and the new target position is ignored.

NOTE
Calling the “posd” function will result in a movement if the drive is not interrupted. The target position will be accepted even when the drive is interrupted.

Block header:

```
Name:          posd
Type:          CONTROLLER_FUN

VAR_INPUT
  posvalue     DINT
  resource     WORD
VAR_END

VAR_OUTPUT
  posd         BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
posvalue:	Position value	Target position in drive units
resource:	x1, x2 ... xn	Axis 1, 2 ... n

Programming example:

```
ld          5000
posd        x1
```

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.61 posdf, Absolute positioning in drive units and wait until position reached

Description:

The manufacturer-defined function “posdf” is used in point-to-point mode to preset an absolute target position in drive units, start a positioning operation and wait until the target position has been reached. The controlling task is stopped during the movement.

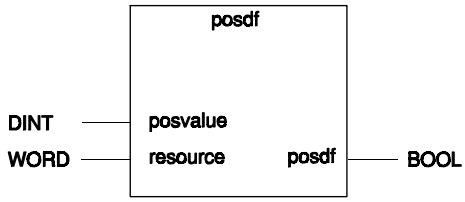
NOTE
The “posdf” function does not return a result until the setpoint has been reached.
If an interruption occurs during a movement, the function is not terminated.

NOTE
The following requirements must be met before specifying a new target position:
- The axis must be in point-to-point mode.
- The acceleration must be defined.
- The axis must not be interrupted.
- The axis must not be "blocked".
If any of these requirements is not met, an error is generated and the new target position is ignored.

Block header:

```
Name:          posdf
Type:          CONTROLLER_FUN

VAR_INPUT
  posvalue     DINT
  resource     WORD
VAR_END

VAR_OUTPUT
  posdf        BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
posvalue:	Position value	Target position in drive units
resource:	x1, x2 ... xn	Axis 1, 2 ... n

Programming example:

```
ld             5000
posdf          x1
```

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.62 posdw, Absolute positioning in drive units and wait for end of movement

Description:

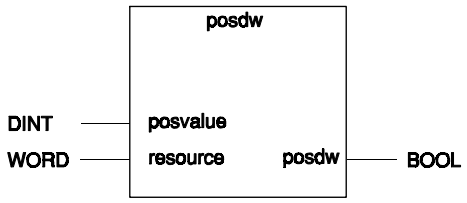
The manufacturer-defined function “posdw” is used in point-to-point mode for presetting an absolute target position in drive units and starting a positioning operation. The program is stopped until the movement has been completed or interrupted. The controlling task is stopped during the movement.

NOTE
The following requirements must be met before specifying a new target position:
- The axis must be in point-to-point mode.
- The acceleration must be defined.
- The axis must not be interrupted.
- The axis must not be "blocked".
If any of these requirements is not met, an error is generated and the new target position is ignored.

Block header:

```
Name:          posdw
Type:          CONTROLLER_FUN

VAR_INPUT
  posvalue     DINT
  resource     WORD
VAR_END

VAR_OUTPUT
  posdw        BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
posvalue:	Position value	Target position in drive units
resource:	x1, x2 ... xn	Axis 1, 2 ... n

Programming example:

```
ld          5000
posdw       x1
```

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE). If the movement is interrupted (e.g. by stop), it returns 0.
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.63 posf, Absolute positioning in user-defined units and wait until position reached

Description:

The manufacturer-defined function “posf” is used in point-to-point mode to preset an absolute target position in user-defined units, start a positioning operation and wait until the target position has been reached. The controlling task is stopped during the movement.

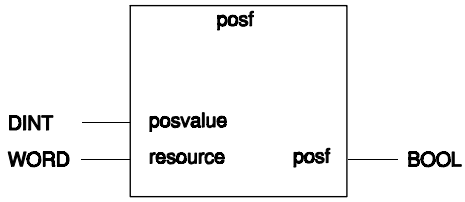
NOTE
The “posf” function does not return a result until the setpoint has been reached.
If an interruption occurs during a movement, the function is not terminated.

NOTE
The following requirements must be met before specifying a new target position:
- The axis must be in point-to-point mode.
- The acceleration must be defined.
- The axis must not be interrupted.
- The axis must not be "blocked".
If any of these requirements is not met, an error is generated and the new target position is ignored.

Block header:

```
Name:          posf
Type:          CONTROLLER_FUN

VAR_INPUT
  posvalue     DINT
  resource     WORD
VAR_END

VAR_OUTPUT
  posf         BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
posvalue:	Position value	Target position in user-defined units
resource:	x1, x2 ... xn	Axis 1, 2 ... n

Programming example:

```
ld      5000
posf    x1
```

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.64 posw, Absolute positioning in user-defined units and wait for end of movement

Description:

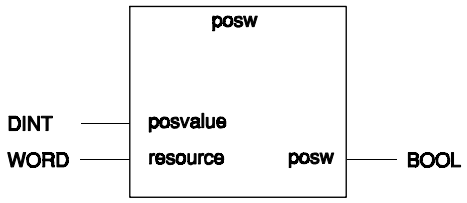
The manufacturer-defined function “posw” is used in point-to-point mode for presetting an absolute target position in user-defined units and starting a positioning operation. The program is stopped until the movement has been completed or interrupted. The controlling task is stopped during the movement.

NOTE
The following requirements must be met before specifying a new target position:
- The axis must be in point-to-point mode.
- The acceleration must be defined.
- The axis must not be interrupted.
- The axis must not be "blocked".
If any of these requirements is not met, an error is generated and the new target position is ignored.

Block header:

```
Name:          posw
Type:          CONTROLLER_FUN

VAR_INPUT
  posvalue     DINT
  resource     WORD
VAR_END

VAR_OUTPUT
  posw         BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
posvalue:	Position value	Target position in user-defined units
resource:	x1, x2 ... xn	Axis 1, 2 ... n

Programming example:

```
ld      5000
posw    x1
```

Function result:

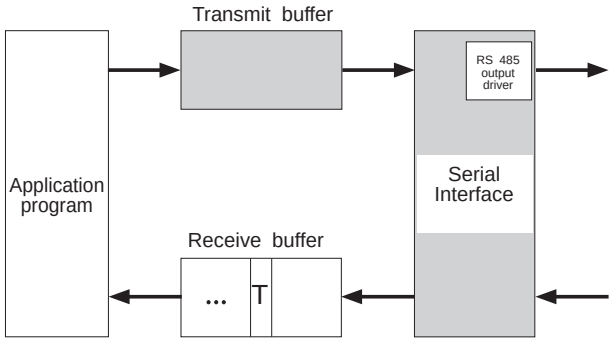
If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE). If the movement is interrupted (e.g. by stop), it returns 0.
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.65 receive_char, Read single character from receive buffer

Description:

The manufacturer-defined function “receive_char” reads single characters from the receive buffer.

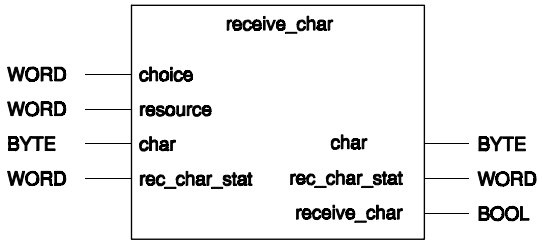
NOTE
The interface must have been initialized with the manufacturer-defined function “com_init_asc”.



Block header:

```
Name:           receive_char
Type:           CONTROLLER_FUN

VAR_INPUT
  choice        WORD
  resource      WORD
VAR_END

VAR_OUTPUT
  receive_char  BOOL
VAR_END

VAR_IN_OUT
  char          BYTE
  rec_char_stat WORD
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	0	Currently not used
resource:	c1, c2	Serial interface 1, 2
char:	-	Contains the read-in character
rec_char_stat:	-	Status word which can be retrieved after the function call using “getstate”; see chapter 7.2.40.

Programming example: ***Comment:***

```
VAR
  char          BYTE        0
  status        WORD        0
VAR_END

ld              0
receive_char    c2, char, status
```

Read character from receive buffer into “char” variable

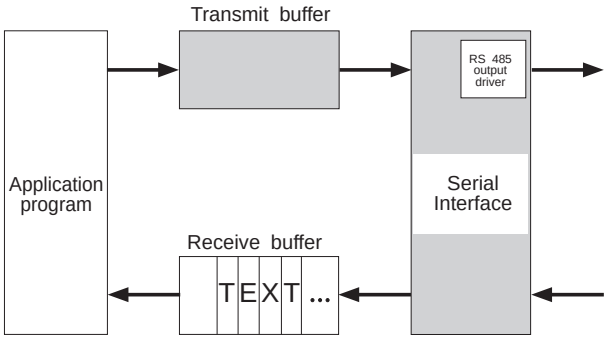
Function result:

If executed successfully, the function returns TRUE, otherwise FALSE. If the function returns FALSE, the cause can be determined from the status returned. You can also use “getstate” to retrieve the status explicitly from another point in the program and retrieve a potential error cause using “geterror_sr”.

7.2.66 receive_string, Read string from transmit and receive buffer

Description:
The manufacturer-defined function “receive_string” reads a string from the receive buffer and stores it in the "string" parameter.
The "end_char" parameter can be used to specify the length of the string by specifying the last character in the receive buffer which will be read in.

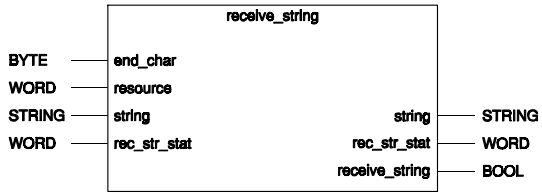
NOTE
The interface must have been initialized with the manufacturer-defined function “com_init_asc”.



Block header:

```
Name:           receive_string
Type:           CONTROLLER_FUN

VAR_INPUT
  end_char      BYTE
  resource      WORD
VAR_END

VAR_OUTPUT
  receive_string  BOOL
VAR_END

VAR_IN_OUT
  string        STRING
  rec_str_stat  WORD
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
end_char:	-	End character of received string
resource:	c1, c2	Serial interface 1, 2
string:	-	Contains the read-in string
rec_str_stat:	-	Status word; see “getstate” in chapter 7.2.40.

Programming example: ***Comment:***

```
VAR
  rec_buf       STRING (100)    '0'
  com_status    WORD             0
VAR_END

ld              16#0D
receive_string  c2, rec_buf, com_status
```

Pass string including end-of-string character from receive buffer to “rec_buf”

Function result:

If executed successfully, the function returns TRUE, otherwise FALSE. If the function returns FALSE, the cause can be determined from the status returned. You can also use “getstate” to retrieve the status explicitly from another point in the program and retrieve a potential error cause using “geterror_sr”.

7.2.67 record, Start recording

Description:

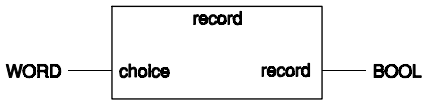
If the controller is set up with the ONLINE3 software, the “record” function can be used to start a recording operation; see ONLINE3 documentation.

Block header:

Name: record
Type: CONTROLLER_FUN

```
VAR_INPUT
  choice        WORD
VAR_END

VAR_OUTPUT
  record        BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	0	Not yet implemented

Programming example:

```
ld          0
record
```

Function result:

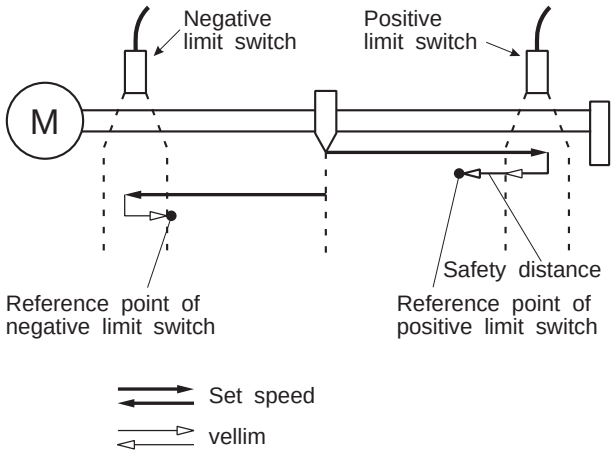
If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).

7.2.68 refpos, Reference movement

Description:

The “refpos” function executes a movement towards a limit or reference switch. The limit or reference switch is selected with “limit”. Reference movements are only possible in point-to-point mode.
When the reference movement is started with “refpos”, the stepping motor approaches the limit switch at the set speed. Subsequently the stepping motor moves in the opposite direction to the reference point at “vellim” speed.

NOTE
You can use “setpos(d)” to define a safety clearance before a reference movement.

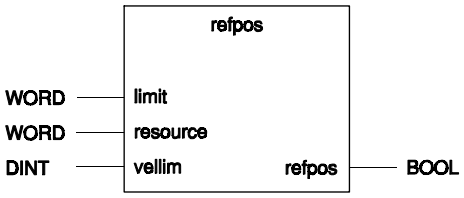


Block header:

Name: refpos
Type: CONTROLLER_FUN

```
VAR_INPUT
  limit      WORD
  resource   WORD
  vellim     DINT
VAR_END

VAR_OUTPUT
  refpos     BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
limit:	limp, limn, ref	limp = Positive limit switch limn = Negative limit switch ref = Reference switch
resource:	x1, x2 ... xn	Axis 1, 2 ... n
vellim:	-	Speed for movement away from reference switch

Programming examples:

```
ld      limp
refpos  x1, 100
```
Move to positive limit switch

```
ld      ref
or      limp
refpos  x1, 200
```
Move to the right to reference switch

```
ld      ref
or      limn
refpos  x1, 200
```
Move to the left to reference switch

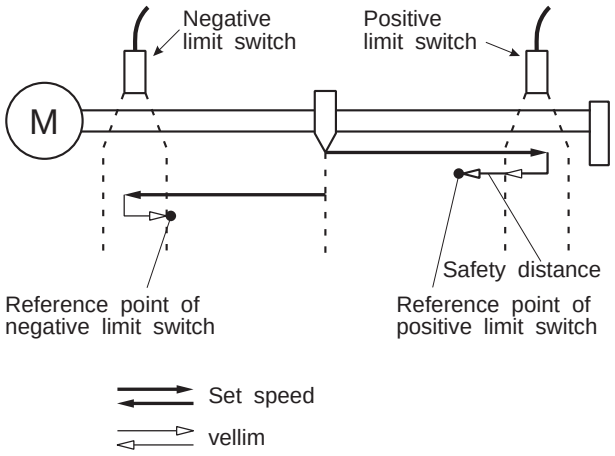
Function result:

If executed successfully, the function returns TRUE, if not, it returns FALSE. The status of the reference movement can be determined with "getstate".

7.2.69 refposf, Reference movement and wait until reference point reached

Description:

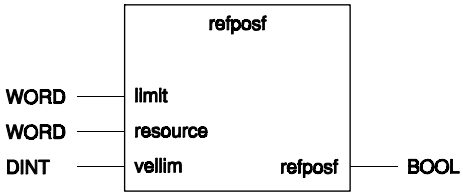
The “refposf” function is identical to the “refpos” function, with one exception: Program execution is suspended until the reference movement has been successfully completed.
If an error should occur during the reference movement, further program execution is blocked by the reference movement.



Block header:

```
Name:          refposf
Type:          CONTROLLER_FUN

VAR_INPUT
  limit        WORD
  resource     WORD
  vellim       DINT
VAR_END

VAR_OUTPUT
  refposf      BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
limit:	limp, limn, ref	limp = Positive limit switch limn = Negative limit switch ref = Reference switch
resource:	x1, x2 ... xn	Axis 1, 2 ... n
vellim:	-	Speed for movement away from reference switch

Programming examples:

```
ld          limp
refposf     x1, 100
```
Move to positive limit switch

```
ld          ref
or          limp
refposf     x1, 200
```
Move to the right to reference switch

```
ld          ref
or          limn
refposf     x1, 200
```
Move to the left to reference switch

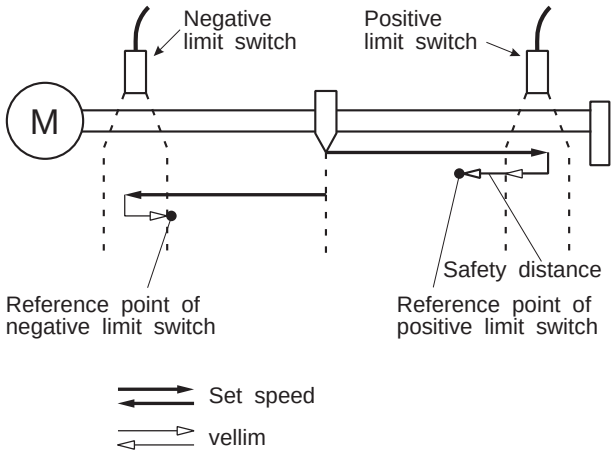
Function result:

If executed successfully, the function returns TRUE, if not, it returns FALSE. The status of the reference movement can be determined with "getstate".

7.2.70 refposw, Reference movement and wait for end of movement

Description:

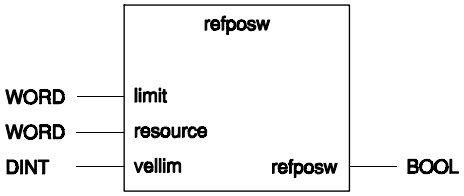
The “refposw” function is identical to the “refpos” function, with one exception: Program execution is suspended until the reference movement has been completed.
The reference movement status can be used to determine whether the reference movement was successful.



Block header:

```
Name:          refposw
Type:          CONTROLLER_FUN

VAR_INPUT
  limit        WORD
  resource     WORD
  vellim       DINT
VAR_END

VAR_OUTPUT
  refposw      BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
limit:	limp, limn, ref	limp = Positive limit switch limn = Negative limit switch ref = Reference switch
resource:	x1, x2 ... xn	Axis 1, 2 ... n
vellim:	-	Speed for movement away from reference switch

Programming examples:		***Comment:***
ld	limp	Move to positive limit switch
refposw	x1, 1000	
ld	ref	
or	limp	Move to the right to reference switch
refposw	x1, 200	
ld	ref	
or	limn	Move to the left to reference switch
refposw	x1, 200	

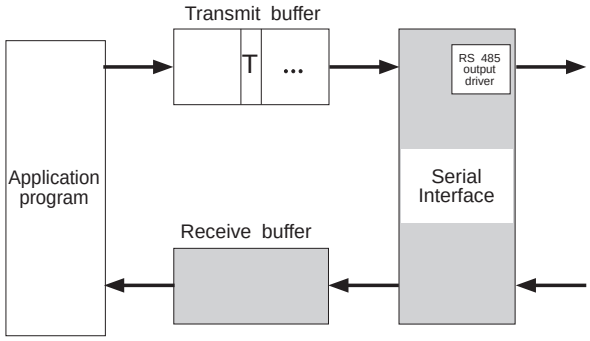
Function result:
If the reference movement is executed successfully, the function returns TRUE, if not, it returns FALSE. The status of the reference movement can be determined with “getstate”.

7.2.71 send_char, Output single characters to serial interface

Description:

The manufacturer-defined function “send_char” writes individual characters into the transmit buffer. The character is then output automatically to a serial interface.

NOTE
The interface must have been initialized with the manufacturer-defined function “com_init_asc”.

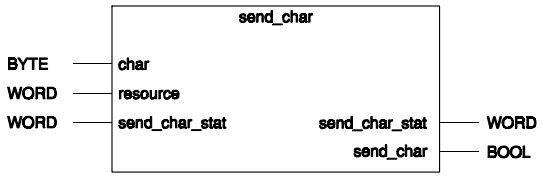


Block header:

Name: send_char
Type: CONTROLLER_FUN

```
VAR_INPUT
  char          BYTE
  resource      WORD
VAR_END

VAR_OUTPUT
  send_char     BOOL
VAR_END

VAR_IN_OUT
  send_char_stat WORD
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
send_char:	-	Character to be output
resource:	c1, c2	Serial interface 1, 2
send_char_stat:	-	Status word; see “getstate” in chapter 7.2.40.

Programming example:

```
VAR
com_status      WORD        0
VAR_END

ld              16#31
send_char       c2, com_status
```

Comment:

Output character ’1’

Function result:

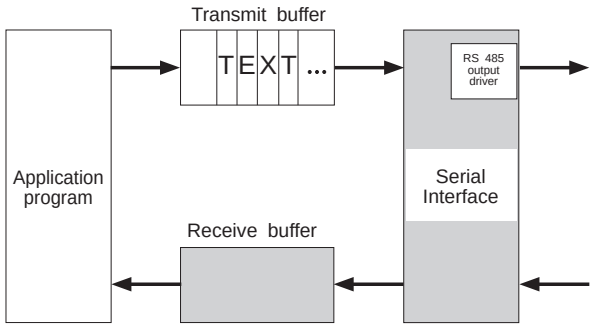
If executed successfully, the function returns TRUE, otherwise FALSE. If the function returns FALSE, the cause can be determined from the status returned. You can also use “getstate” to retrieve the status explicitly from another point in the program and retrieve a potential error cause using “geterror_sr”.

7.2.72 send_string, Output string to serial interface

Description:

The manufacturer-defined function “send_string” writes a string into the transmit buffer. The string is then output automatically to a serial interface.

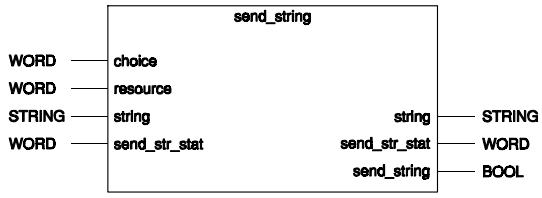
NOTE
The interface must have been initialized with the manufacturer-defined function “com_init_asc”.



Block header:

```
Name:           send_string
Type:           CONTROLLER_FUN

VAR_INPUT
  choice        WORD
  resource      WORD
VAR_END

VAR_OUTPUT
  send_string   BOOL
VAR_END

VAR_IN_OUT
  string        STRING
  send_str_stat WORD
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice	0	Currently not used
resource:	c1, c2	Serial interface 1, 2
string:	-	String to be sent
send_string_stat	-	Status word; see “getstate” in chapter 7.2.40

Programming example: ***Comment:***

```
VAR
  string        STRING (4)      'TEXT'
  status        WORD            0
VAR_END

ld              0
send_string     c2, string, status      Output 'TEXT' string
```

Function result:

If executed successfully, the function returns TRUE, otherwise FALSE. If the function returns FALSE, the cause can be determined from the status returned. You can also use “getstate” to retrieve the status explicitly from another point in the program and retrieve a potential error cause using “geterror_sr”.

7.2.73 setaccsig, Signal-dependent deceleration

Description:

The manufacturer-defined function “setaccsig” can be used to link signals (e.g. stop) to emergency deceleration ramps. When the specified signal is activated, the associated emergency deceleration ramp is selected.

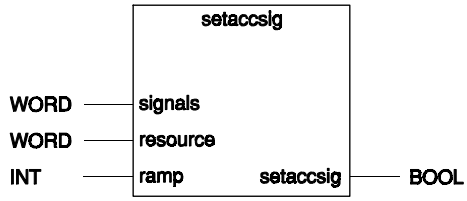
NOTE
The following requirements must be met prior to selecting a new ramp:
- *The axis must be at a standstill or inactive.*
- *The ramp storage number must be in the range from 1 to 10.*
- *The ramp stored under the specified storage number must have been calculated with the manufacturer-defined function “calcaccel”.*

NOTE
After deceleration with an emergency deceleration ramp, the ramp selected with the manufacturer-defined function “accel” is reloaded. The emergency deceleration ramp must be steeper than the acceleration ramp. The link between a ramp and a signal can be cancelled by passing the “no_sig” constant as the signal parameter.

Block header:

```
Name:          setaccsig
Type:          CONTROLLER_FUN

VAR_INPUT
  signals      WORD
  resource     WORD
  ramp         INT
VAR_END

VAR_OUTPUT
  setaccsig    BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
signals:	limp	Positive hardware limit switch
	limn	Negative hardware limit switch
	ref	Reference switch
	stop	Hardware stop input
	swstop	Software stop
	dragerr	Contouring error
resource:	x1, x2 ... xn	Axis 1, 2 ... n
ramp	1 - 10	Storage number 1 - 10

Programming example:		***Comment:***
ld	limp	Ramp no. 10 is used for decelerating
setaccsig	x1, 10	at the positive limit switch.

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.74 setanalog, Write analog value to an output channel of an analog module

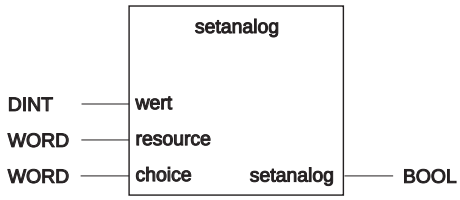
Description:

This function can be used to write an analog value to an output channel of an analog module.

Block header:

```
Name:          setanalog
Type:          CONTROLLER_FUN

VAR_INPUT
  value        DINT
  resource     WORD
  choice       WORD
VAR_END

VAR_OUTPUT
  setanalog    BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
value:	DINT	Output voltage value in mV
resource:	ad1	Analog value for slot 51
	ad2	Analog value for slot 53
choice:	1	Number of analog output on analog module

Programming example:

Write a voltage value of 5.2 V to the first analog output on the analog module in slot 51

```
ld          5200
setanalog   ad1, 1
```

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).

7.2.75 setcurrent, Set electrical current values for various movement states

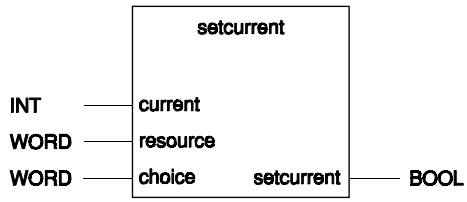
Description:

The manufacturer-defined function “setcurrent” can be used for setting electrical current values in the range from 0 to 100% of the set phase current for the various movement stages of a drive.
Three programmable electrical current ranges are available for the motor current control. The electrical current ranges are specified as a percentage of the maximum current. The maximum current is set on the power electronics and depends on the motor.
The electrical current ranges take effect in the various movement stages. A current value can be specified for axis standstill, for the acceleration and deceleration phase as well as for the constant movement phase.

Block header:

```
Name:          setcurrent
Type:          CONTROLLER_FUN

VAR_INPUT
  current      INT
  resource     WORD
  choice       WORD
VAR_END

VAR_OUTPUT
  setcurrent   BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
current:	0 - 100	Electrical current values as a percentage
resource:	x1, x2 ... xn	Axis 1, 2 ... n
choice:	stand	Current for standstill
	accel	Current for acceleration and deceleration phases
	constant	Current range for constant movement

Programming example:		***Comment:***
ld	50	Set 50% of maximum current for
setcurrent	x1, stand	the standstill phase

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.76 setdrive, Set encoder input for position feedback

Description:

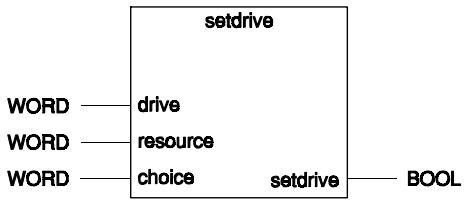
The manufacturer-defined function “setdrive” assigns an encoder for motor actual position detection (e.g. for rotation monitoring) to a drive axis.

NOTE
An encoder used as a source for an external reference variable (in position following mode) may not be assigned with the manufacturer-defined function “setdrive”.

Block header:

```
Name:          setdrive
Type:          CONTROLLER_FUN

VAR_INPUT
  drive        WORD
  resource     WORD
  choice       WORD
VAR_END

VAR_OUTPUT
  setdrive     BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
drive:	x1	Axis
resource:	p1, p2	Encoder 1, encoder 2
choice:	0, ≠ 0	= 0 Cancel link ≠ 0 Link encoder to axis

Programming example:

```
ld          x1
setdrive    p1, 1
```

Comment:

Axis 1 is monitored by encoder 1

Function result:

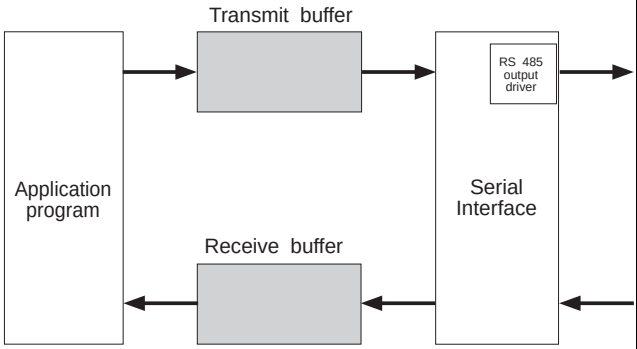
If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.20.

7.2.77 sethardware, Hardware settings for serial interface

Description:

The manufacturer-defined function “sethardware” can be used for selecting hardware settings. In this particular case it is used for enabling or disabling the RS 485 interface driver for a serial interface.

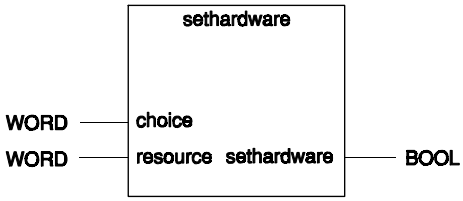
NOTE
The “sethardware” function can also be used for setting various axis parameters. For more information, see chapter 7.2.78.



Block header:

Name:	sethardware
Type:	CONTROLLER_FUN
VAR_INPUT	
choice	WORD
resource	WORD
VAR_END	
VAR_OUTPUT	
sethardware	BOOL
VAR_END	

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	txd_on	Enable RS 485 interface driver
	txd_off	Disable RS 485 interface driver
resource:	c1, c2	Serial interface 1, 2
	(x1, x2 ... xn)	(Axis 1, 2 ... n; see chapter 7.2.78)

Programming example:

ld	txd_on
sethardware	c2

Function result:

If executed successfully, the function returns TRUE, if not, it returns FALSE.

7.2.78 sethardware, Hardware settings for axes

Description:

The manufacturer-defined function “sethardware” can be used for selecting hardware settings.

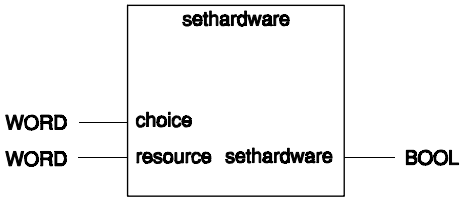
NOTE
Axis-related hardware settings can only be changed when the axis is at a standstill.

NOTE
The “sethardware” function can also be used for enabling or disabling the RS 485 interface driver for a serial interface. For more information, see chapter 7.2.77.

Block header:

```
Name:          sethardware
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
VAR_END

VAR_OUTPUT
  sethardware  BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	pulson	Switch on pulse output to power controller
	pulsoff	Switch off pulse output to power controller
	ampon	Enable power controller
	ampoff	Disable power controller
	dirinvert [1)]	Invert sense of rotation
	trig_off	Disable trigger input of signal interface
	trig_on	Enable trigger input of signal interface
	dirnormal [1)]	Set default sense of rotation
		NOTE *If rotation monitoring is enabled, the sense of rotation may not be inverted.*
resource:	x1, x2 ... xn (c1, c2)	Axis 1, 2 ... n (Serial interface 1, 2; see chapter 7.2.77)

Programming example:		***Comment:***
ld	ampon	Issue power controller enable
sethardware	x1	

Function result:

If executed successfully, the function returns TRUE, otherwise FALSE.
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

1) Not available for units with internal position controller: e.g. WDP3-337 and WDP3-338

7.2.79 setmode, Set axis operating mode

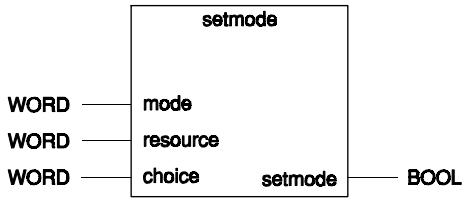
Description:

The manufacturer-defined function “setmode” sets the operating mode of an axis or an encoder (see chapter 7.2.80). This is only admissible when the axis is at a standstill. At power-on, the controller defaults to
point-to-point mode.

Block header:

```
Name:          setmode
Type:          CONTROLLER_FUN

VAR_INPUT
  mode         WORD
  resource     WORD
  choice       WORD
VAR_END

VAR_OUTPUT
  setmode      BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
mode:	ptp	Point-to-point mode
	velocity	Speed mode
	pos_drag	Position following mode
resource:	x1, x2 ... xn	Axis 1, 2 ... n
	(p1, p2)	(Encoder 1, encoder 2; see chapter 7.2.80)
choice:	drive	Indexer

Programming example:		***Comment:***
ld	ptp	Set point-to-point mode for axis x1
setmode	x1, drive	
ld	velocity	Set speed mode for axis x1
setmode	x1, drive	

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.80 setmode, Set encoder operating mode

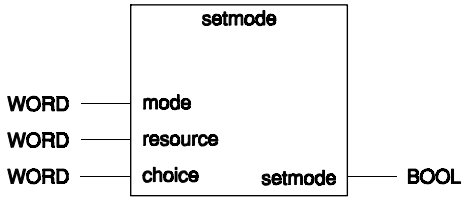
Description:

The manufacturer-defined function “setmode” sets the operating mode of an axis (see 7.2.79) or an encoder.

Block header:

```
Name:          setmode
Type:          CONTROLLER_FUN

VAR_INPUT
  mode         WORD
  resource     WORD
  choice       WORD
VAR_END

VAR_OUTPUT
  setmode      BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
mode: (for choice=drive)	watchdrive [1)] none [1)]	Activate rotation monitoring No assignment (factory setting)
mode: (for choice=encoder)	encquad encdouble encsingle encpulsdir indexoff indexstop indextrigger	4x evaluation of encoder signals 2x evaluation of encoder signals 1x evaluation of encoder signals (factory setting) Pulse and direction signal acquisition Normal index pulse evaluation Index pulse as stop signal Index pulse as trigger signal
resource:	p1, p2 (x1, x2 ... xn)	Encoder 1 [1)], encoder 2 (Axis 1, 2 ... n; see chapter 7.2.79)
choice:	drive, encoder	Indexer, encoder

Programming example:		***Comment:***
ld setmode	encquad p2, encoder	Set encoder 2 to quadruple evaluation

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.20.

1) Not available for units with internal position controller: e.g. WDP3-337 and WDP3-338

7.2.81 setnorm, Set axis normalizing factors

Description:

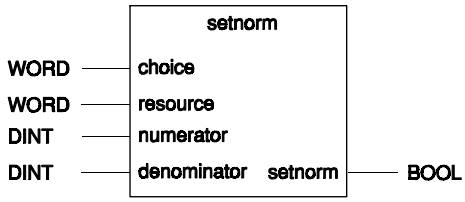
The manufacturer-defined function “setnorm” can be used to set normalizing factors or gear ratios. The "numerator" parameter specifies the numerator, the "denominator" parameter specifies the denominator for the normalizing operation. The "choice" parameter is used for selecting the desired normalizing factors.

NOTE
Specifying 0 for the numerator or denominator is invalid and will cause an error. However, the numerator 0 is admissible for reference variable normalization (i.e. external reference variable = 0).

Block header:

```
Name:          setnorm
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
  numerator    DINT
  denominator  DINT
VAR_END

VAR_OUTPUT
  setnorm      BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	position	Position normalizing
	velocity	Speed normalizing
	accel	Acceleration normalizing
	source	Reference variable normalizing (gear ratio)
resource:	x1, x2 ... xn	Axis 1, 2 ... n
	(p1, p2)	(Encoder 1, encoder 2; see chapter 7.2.82)
numerator:	value	Numerator of normalizing factor
denominator:	value	Denominator of normalizing factor

Programming example:		***Comment:***
ld	position	Set normalizing factor for positions
setnorm	x1, 100, 1	to 100 : 1

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.82 setnorm, Set encoder normalizing factors

Description:

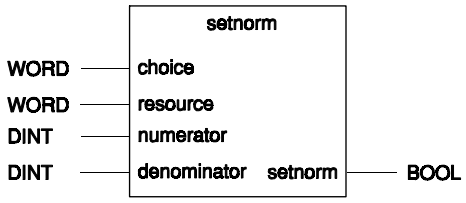
The manufacturer-defined function “setnorm” is used to set normalizing factors. This enables you to perform conversions among user-defined units (positioning commands), drive units (step resolution of the corresponding stepping motor, e.g. 1000 steps per 360°) and encoder units (pulse resolution of the connected incremental encoder, e.g. 4000 increments per 360°). The "numerator" parameter specifies the numerator, the "denominator" parameter specifies the denominator for the normalizing operation. The "choice" parameter is used for selecting the desired normalizing factors.

NOTE
Specifying 0 for the numerator or denominator is invalid.

Block header:

```
Name:           setnorm
Type:           CONTROLLER_FUN

VAR_INPUT
  choice        WORD
  resource      WORD
  numerator     DINT
  denominator   DINT
VAR_END

VAR_OUTPUT
  setnorm             BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	position	Drive unit to user-defined unit ratio $Drive\ unit = User\text{–}defined\ unit\ x\ \frac{Numerator}{Denominator}$
	indexer	Encoder unit to drive unit ratio $Encoder\ unit = Driveunit\ x\ \frac{Numerator}{Denominator}$
resource:	p1, p2 (x1, x2 ... xn)	Encoder 1, encoder 2 (Axis 1, 2 ... n; see chapter 7.2.81)
numerator:	value	Numerator of normalizing factor
denominator:	value	Denominator of normalizing factor

Programming example:

```
ld          position
setnorm     p1, 100, 1
```

Function result:

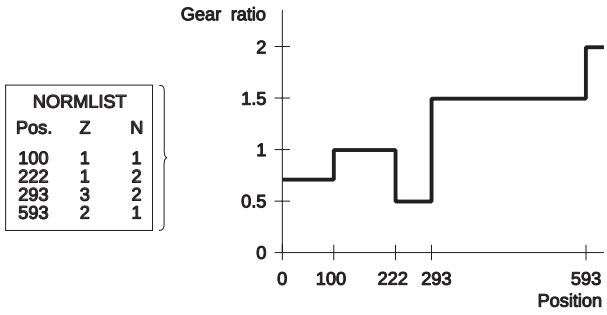
If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.20.

7.2.83 setnormlist, Enable position list for gear ratios

Description:

The “setnormlist” function is used for setting a position normalizing list. The position normalizing list contains position specifications. Each position is assigned a normalizing factor (gear ratio). When passing these positions, the associated normalizing factor is activated.
When a position list is processed, no other position list can be set.

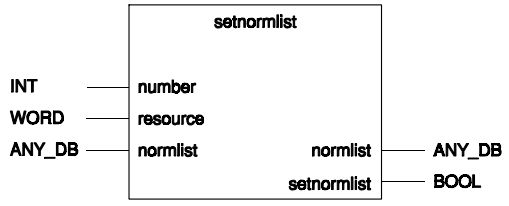
NOTE
The positions are absolute positions expressed in drive units.
“setnormlist” cannot be activated while “setvellist” or “setsiglist” are active.



Block header:

```
Name:           setnormlist
Type:           CONTROLLER_FUN

VAR_INPUT
  number        INT
  resource      WORD
VAR_END

VAR_OUTPUT
  setnormlist   BOOL
VAR_END

VAR_IN_OUT
  normlist      ANY_DB
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
number:	-	Specifies the number of parameters to be processed from the list
resource:	x1, x2 ... xn	Axis 1, 2 ... n
normlist:	-	Specifies the data block containing the list

Programming example:

```
ld              4
setnormlist     x1, dbnormlist
```

Function result:

If executed successfully, the function returns TRUE, if not, it returns FALSE.

7.2.84 setparam, Set control parameters

Description:

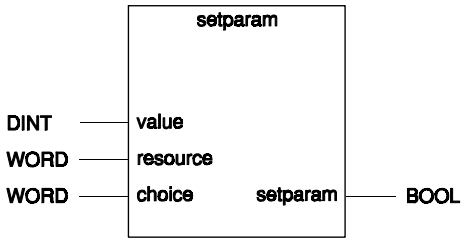
The “setparam” function can be used to enter the control parameters of the PID position controller.

NOTE
The “setparam” function is only applicable to series 300 units with PID position controller (e.g. WDP3-337, WDP3-338). The control parameters are explained in the respective controller manual.

Block header:

```
Name:           setparam
Type:           CONTROLLER_FUN

VAR_INPUT
  value         DINT
  resource      WORD
  choice        WORD
VAR_END

VAR_OUTPUT
  setparam      BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
value:	0 < kp < 16383 0 < ki < 16383 -16383 < kd < 16383 -16383 < kf < 16383 -16383 < ka < 16383 -2147483648 < offset < 2147483648 ta = 1, 2, 3, 4 (ms)	Value of selected parameter; see choice
resource:	x1	Axis
choice:	kp	Proportional component
	ki	Integral component
	kd	Differential component
	kf	Speed offset
	ka	Acceleration offset
	offset	Manipulated variable offset
	ta	Sampling time

Programming example:		***Comment:***
ld	100	Set proportional component kp
setparam	x1, kp	of axis x1 to 100

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.85 setpos, Set axis positions in user-defined units

Description:

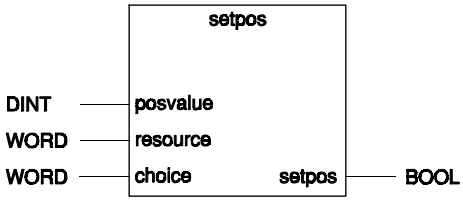
The manufacturer-defined function “setpos” can be used for setting fixed axis positions in user-defined units. The user-defined units are converted to drive units using the position normalizing factors. The "choice" parameter specifies the position value to be set.

NOTE
Setting dimensions and setting the positive or negative software limit switch is only permitted at standstill. The current position must be within the software limit switch range.

Block header:

```
Name:          setpos
Type:          CONTROLLER_FUN

VAR_INPUT
  posvalue     DINT
  resource     WORD
  choice       WORD
VAR_END

VAR_OUTPUT
  setpos       BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
posvalue:	Position value	Positions in user-defined units
resource:	x1, x2 ... xn (p1, p2)	Axis 1, 2 ... n (Encoder 1, encoder 2; see chapter 7.2.86)
choice:	actual	Actual position or setting dimensions (a position overrun is cleared)
	swlimp	Value of positive software limit switch (the positive limit switch value must be greater than the negative limit switch value)
	swlimn	Value of negative software limit switch (the negative limit switch value must be less than the positive limit switch value)
	srcoffset	Reference variable offset in position following mode
	refsecure	Safety distance from limit switch or reference switch after a reference movement
	srcdiff	Indexer following error
	maxdragerr	Resetting the greatest contouring error which occurred (see “getpos”)
	jump [1)]	Apply positional jump (absolute position) to input of position controller

Programming example:

```
ld             1000
setpos         x1, actual
```

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

1) Only available for units with internal position controller: e.g. WDP3-337 and WDP3-338

7.2.86 setpos, Set encoder positions in user-defined units

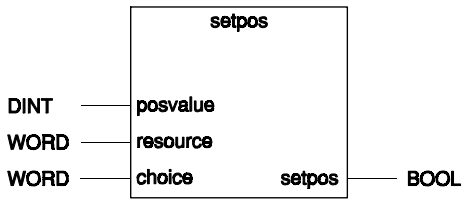
Description:

The manufacturer-defined function “setpos” can be used for setting encoder positions in user-defined units. This enables you to set the encoder to any value. This process involves a conversion of the specified position value from user-defined units to drive units and subsequently from drive units to encoder units, using the normalizing factors. The "choice" parameter specifies the position value to be set.

Block header:

```
Name:          setpos
Type:          CONTROLLER_FUN

VAR_INPUT
  posvalue     DINT
  resource     WORD
  choice       WORD
VAR_END

VAR_OUTPUT
  setpos       BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
posvalue:	Position value	Positions in user-defined units
resource:	p1, p2 (x1, x2 ... xn)	Encoder 1, encoder 2 (Axis 1, 2 ... n; see chapter 7.2.85)
choice:	actual maxveldrag minveldrag	Set the current encoder position Set the contouring error limit at maximum speed Set the contouring error limit at standstill

Programming example:

```
ld          1000
setpos      p1, actual
```

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.20.

7.2.87 setposd, Set axis positions in drive units

Description:

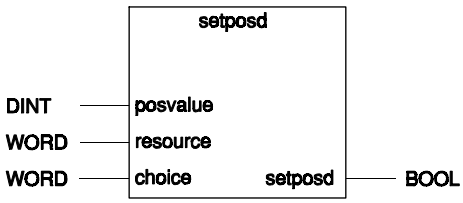
The manufacturer-defined function “setposd” sets fixed axis positions in drive units. Therefore, no normalizing is involved. The "choice" parameter specifies the position value to be set.

NOTE
Setting dimensions and setting the positive or negative software limit switch is only permitted at standstill. The current position must be within the software limit switch range.

Block header:

```
Name:          setposd
Type:          CONTROLLER_FUN

VAR_INPUT
  posvalue     DINT
  resource     WORD
  choice       WORD
VAR_END

VAR_OUTPUT
  setposd      BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
posvalue:	Position value	Positions in drive units
resource:	x1, x2 ... xn (p1, p2)	Axis 1, 2 ... n (Encoder 1, encoder 2; see chapter 7.2.88)
choice:	actual	Actual position or setting dimensions (a position overrun is cleared)
	swlimp	Value of positive software limit switch (the positive limit switch value must be greater than the negative limit switch value)
	swlimn	Value of negative software limit switch (the negative limit switch value must be less than the positive limit switch value)
	srcoffset	Refernce variable offset in position following mode
	refsecure	Safety distance from limit switch or reference switch after a reference movement
	srcdiff	Indexer following error
	maxdragerr	Reset the greatest contouring error which occurred (see “getpos”)
	jump [1)]	Apply positional jump (absolute position) to input of position controller

Programming example:

```
ld          1000
setposd     x1, actual
```

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

1) Only available for units with internal position controller: e.g. WDP3-337 and WDP3-338

7.2.88 setposd, Set encoder positions in encoder units

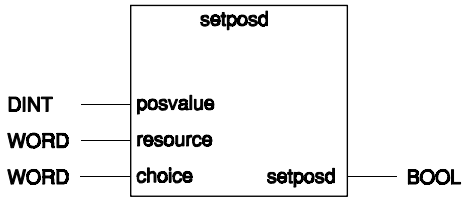
Description:

The manufacturer-defined function “setposd” can be used for setting encoder positions directly in encoder units.
The "choice" parameter specifies the position value to be set.

Block header:

```
Name:          setposd
Type:          CONTROLLER_FUN

VAR_INPUT
  posvalue     DINT
  resource     WORD
  choice       WORD
VAR_END

VAR_OUTPUT
  setposd      BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
posvalue:		Positions in encoder units
resource:	p1, p2 (x1, x2 ... xn)	Encoder 1, encoder 2 (Axis 1, 2 ... n; see chapter 7.2.85)
choice:	actual maxveldrag minveldrag	Set the current encoder position Set the contouring error limit at maximum speed Set the contouring error limit at standstill

Programming example:

```
ld          1000
setposd     p1, actual
```

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.20.

7.2.89 setsig_activ_h, Set active state of axis signals

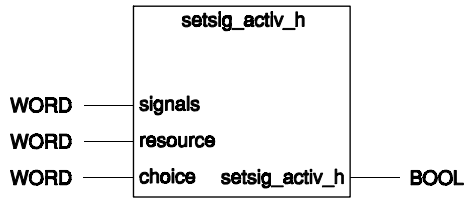
Description:

The manufacturer-defined function “setsig_activ_h” sets the active states of the predefined signal inputs. If the bit position of the corresponding signal is 0, it is interpreted to be active when the input is deenergized. If the bit position is 1 the signal is interpreted to be active when the input is energized.
For an active signal, the corresponding signal flag is set to 1. The signal flags can be retrieved using the manufacturer-defined function “getsig_sr”. The manufacturer-defined function “getsig” retrieves the signals from the inputs (energized = 1, deenergized = 0).

Block header:

```
Name:           setsig_activ_h
Type:           CONTROLLER_FUN

VAR_INPUT
  signals       WORD
  resource      WORD
  choice        WORD
VAR_END

VAR_OUTPUT
  setsig_activ_h  BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
signals:		Active state of signals:
	limp	Positive hardware limit switch
	limn	Negative hardware limit switch
	ref	Reference switch
	trig	Hardware trigger
resource:	x1, x2 ... xn	Axis 1, 2 ... n
choice:	watch	Monitoring signals

Programming example:

```
ld              limp
or              limn
setsig_activ_h  x1, watch
```

Comment:

All signals specified in the OR operation are interpreted to be activ_high.

Function result:

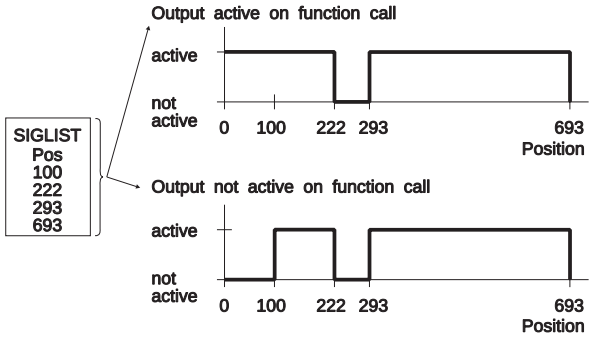
If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.90 setsiglist, Enable position list for signal output

Description:

Enable a position signal list. A position signal list is a data block containing a list of positions. These positions refer to the axis specified by "resource". Whenever the axis passes one of the positions in the list, the signal level of an output changes. If the output was active, it is deactivated, and vice versa. The first position value processed by “setsiglist” will always set the output to active state, regardless of whether it was active or not at the time of the function call.

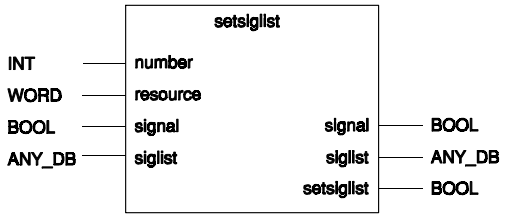
NOTE
The positions are absolute positions expressed in drive units.
The output must be addressed indirectly (e.g. %QX0.1).
“setsiglist” cannot be activated in parallel to “setvellist” and “setnormlist”.



Block header:

```
Name:           setsiglist
Type:           CONTROLLER_FUN

VAR_INPUT
  number        INT
  resource      WORD
VAR_END

VAR_OUTPUT
  setsiglist    BOOL
VAR_END

VAR_IN_OUT
  signal        BOOL
  siglist       ANY_DB
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
number:	-	Specifies the number of positions to be processed
resource:	x1, x2 ... xn	Axis 1, 2 ... n
signal:	-	Specifies the output
siglist:	-	Data block containing the position list

Programming example:

```
ld              4
setsiglist      x1, %QX0.0, dbsiglist
```

Function result:

If executed successfully, the function returns TRUE, if not, it returns FALSE.

7.2.91 setsrcres, Set encoder input for position following mode

Description:

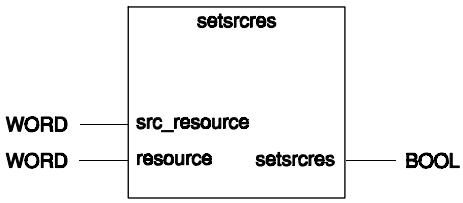
The manufacturer-defined function “setsrcres” selects an encoder as the external reference variable. This allows several encoders to be defined as reference variables. As a prerequisite, the manufacturer-defined function “setsrctyp” must have been used to define the source type “srcenc”.

NOTE
The “setsrcres” function must be called in either point-to-point or speed operating mode.

Block header:

```
Name:          setsrcres
Type:          CONTROLLER_FUN

VAR_INPUT
  src_resource WORD
  resource     WORD
VAR_END

VAR_OUTPUT
  setsrcres    BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
src_resource:	p1, p2	Encoder 1, Encoder 2
resource:	x1	Achse

Programming example:		***Comment:***
ld	p1	Encoder 1 is to be used as the reference variable source for the electronic gear
setsrcres	x1	

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.20.

7.2.92 setsrctime, Set sampling time for position following mode

Description:

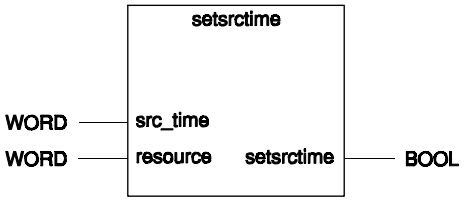
The manufacturer-defined function “setsrtime” controls the sampling time of the external reference variable for position following mode. The greater this value, the lower is the system load. The time is specified in milliseconds.

NOTE
The sampling time set with “setsrctime” is activated upon changing to position following mode (see “setmode” function).

Block header:

```
Name:          setsrctime
Type:          CONTROLLER_FUN

VAR_INPUT
  src_time     TIME
  resource     WORD
VAR_END

VAR_OUTPUT
  setsrctime   BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
src_time:	value	Sampling time in ms
resource:	x1, x2 ... xn	Axis 1, 2 ... n

Programming example:

```
ld             T#2 ms
setsrctime     x1
```

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.93 setsrctyp, Set reference variable type for position following mode

Description:

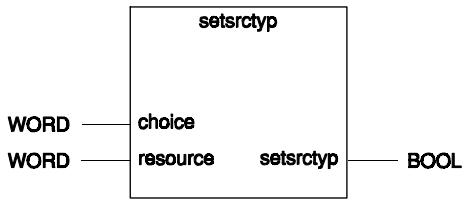
The manufacturer-defined function “setsrctyp” can be used to select the source type (encoder or variable) of the external reference variable for position following mode.

NOTE
The “setsrctyp” function must be called in either point-to-point or speed axis operating mode.

Block header:

```
Name:          setsrctyp
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
VAR_END

VAR_OUTPUT
  setsrctyp    BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	Explanation:
choice:	srcenc	The reference variable source is an encoder
	srcvar	The reference variable source is a variable
	srcnone	No source
resource:	x1, x2 ... xn	Axis 1, 2 ... n

Programming example:

```
ld          srcenc
setsrctyp   x1
```

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.94 setsrcvar, Set variable for position following mode

Description:

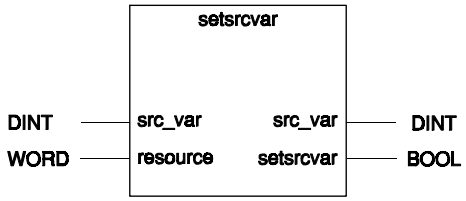
The manufacturer-defined function “setsrcvar” selects a variable as the external reference variable. The value of the variable is used as the setpoint for the drive. A drive setpoint is specified simply by writing a value into this variable. As a prerequisite, the manufacturer-defined function “setsrctyp” must have been used to define the source type “srcvar”.

NOTE
The “setsrcvar” function must be called in either point-to-point or speed operating mode.

Block header:

```
Name:          setsrcvar
Type:          CONTROLLER_FUN

VAR_INPUT
  resource     WORD
VAR_END

VAR_OUTPUT
  setsrcvar    BOOL
VAR_END

VAR_IN_OUT
  src_var      DINT
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
src_var:	variable	Name of variable
resource:	x1, x2 ... xn	Axis 1, 2 ... n

Programming example:

```
VAR
  Variable1    DINT
VAR_END

ld             x1
setsrcvar      variable1
```

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.95 setvel, Set start and maximum speeds

Description:

The manufacturer-defined function “setvel” is used for setting the start and maximum system speeds for an axis. The start speed is the speed at which a positioning operation is started in point-to-point mode. It must be a positive value and less than the maximum system speed.
The maximum system speed is the limit value for speed data. It may only be modified at standstill or with the drive inactive.

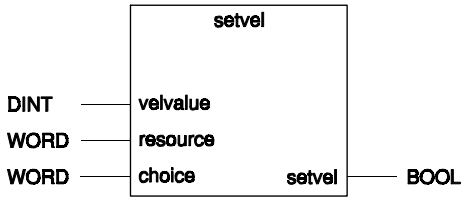
ATTENTION
When modifying the maximum system speed, the acceleration curves must be recalculated using the manufacturer-defined function “calcaccel”, and an acceleration curve must be selected using “accel”.

ATTENTION
If a position/speed list is active (see “setvellist”), the maximum system speed must not be changed.

Block header:

```
Name:          setvel
Type:          CONTROLLER_FUN

VAR_INPUT
  velvalue     DINT
  resource     WORD
  choice       WORD
VAR_END

VAR_OUTPUT
  setvel       BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
velvalue:	0 < Value < Max. system speed 0 < Value	Start speed Maximum system speed
resource:	x1, x2 ... xn	Axis 1, 2 ... n
choice:	start system	Start speed Maximum system speed

Programming example:

```
ld             200
setvel         x1, start
```

Function result:

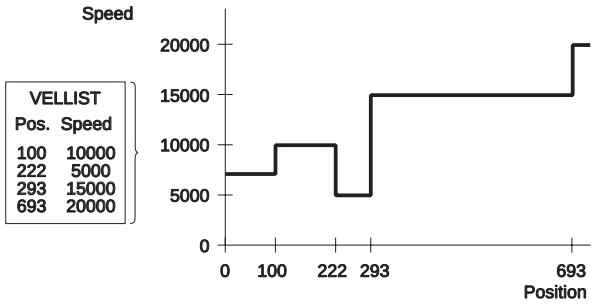
If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.96 setvellist, Enable position list for speed

Description:

The “setvellist” function activates a position/speed list. The position/speed list is a data block containing several positions with associated speeds. These positions refer to the axis specified by "resource". Whenever the axis reaches one of the positions in the list, the speed changes.

NOTE
The positions are absolute positions expressed in drive units.
“setvellist” cannot be activated while “setsiglist” or “setnormlist” are active.



Block header:

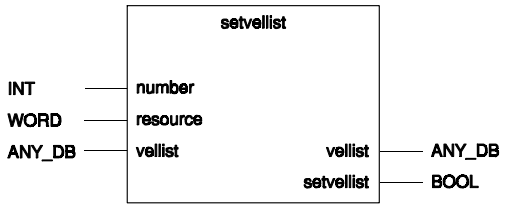
Name: setvellist
Type: CONTROLLER_FUN

VAR_INPUT
  number INT
  resource WORD
VAR_END

VAR_OUTPUT
  setvellist BOOL
VAR_END

VAR_IN_OUT
  vellist ANY_DB
VAR_END

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
number:	-	Specifies the number of positions, or speeds, which are to be processed
resource:	x1, x2 ... xn	Axis 1, 2 ... n
vellist:	-	Data block containing the position/speed list

Programming example:

```
ld          4
setvellist  x1, dbvellist
```

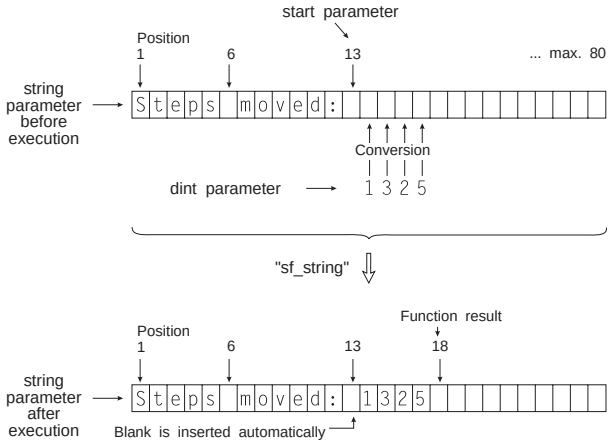
Function result:

If executed successfully, the function returns TRUE, if not, it returns FALSE.

7.2.97 sf_dint, Convert DINT to STRING and insert into string

Description:

The “sf_dint” function converts a number of DINT data type to STRING data type. The converted integer number is appended to a string at the specified "start" position.



Block header:

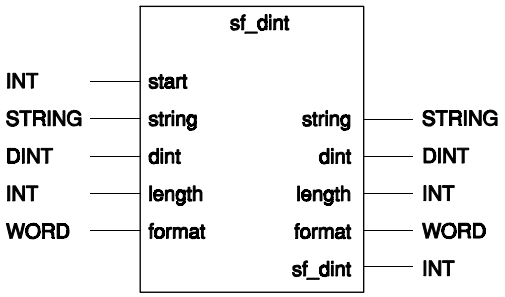
Name: sf_dint
Type: CONTROLLER_FUN

VAR_INPUT
  start INT
VAR_END

VAR_OUTPUT
  sf_dint INT
VAR_END

VAR_IN_OUT
  string STRING
  dint DINT
  length INT
  format WORD
VAR_END

FBD representation



Parameter(s):	***Explanation:***
start:	Specifies the starting position in STRING where the converted DINT number is to be appended. If there is no leading blank before the start position, the “sf_dint” function inserts it automatically (except if start = 1). If the start value is greater than the number of characters in string, the “sf_dint” function pads the space with blanks.
string:	Contains a string. The converted DINT number is appended to this string at the start position. If "string" contains more characters than specified in "start", the remaining characters are truncated.
dint:	Number of DINT type. The number is converted to a string and appended to STRING, beginning at the start position.
length:	Specifies the length for the converted string. If length = 0, the string is formatted to the length determined by the actual number of places. If the number of required places is greater than the length value, “sf_dint” returns the function result -1.

Parameter(s):	***Value/constant:***	***Explanation:***
format:	form_std	Default format: right-justified, no leading zero padding, decimal format
	form_left	Left-justified
	form_0	Leading zero padding
	form_2	Binary formatting
	form_16	Hex formatting
	form_nosp	No leading blank generation
	form_sign	Always generate preceding sign: also +

The system constants consist of a bit pattern. One bit is available for each formatting option. To use several formatting options simultaneously, you can combine the system constants in an OR operation.

Example:

```
ld      form_left
or      form_0
or      form_16
st      form_strg
```

If conflicting formatting options are specified, the function returns -1.

Programming example:

```
VAR
  string        STRING
  dint          DINT
  length        INT
  format        WORD
VAR_END

ld              20
sf_dint         string, dint, length, format
```

Function result:

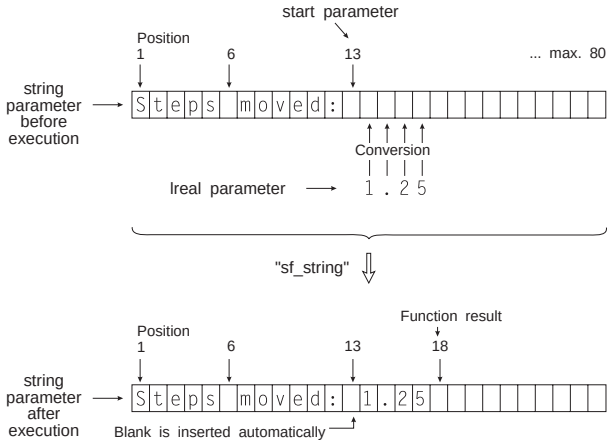
If executed successfully, the function returns the position where it finished the operation. This supplies the new start value for a subsequent string formatting function. If an error occurs, the function returns -1. This would result in a start value of less than 0 for a subsequent string formatting function. In this case, the string formatting function decrements and returns the start value.

NOTE
This is valid for all string functions.

7.2.98 sf_lreal, Convert LREAL to STRING and insert into string

Description:

The “sf_lreal” function converts a number of LREAL data type to STRING data type. The converted number is appended to a string at the specified "start" position.



Block header:

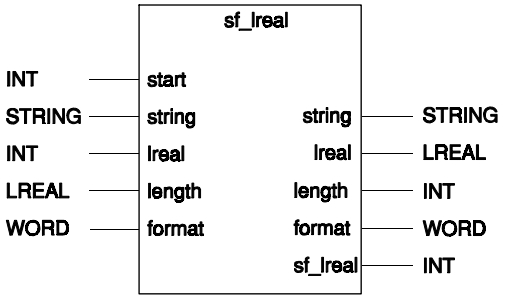
Name: sf_lreal
Type: CONTROLLER_FUN

VAR_INPUT
  start INT
VAR_END

VAR_OUTPUT
  sf_lreal INT
VAR_END

VAR_IN_OUT
  string STRING
  lreal LREAL
  length INT
  format WORD
VAR_END

FBD representation



Parameter(s):	***Explanation:***
start:	Specifies the starting position in STRING where the converted LREAL number is to be appended. If there is no leading blank before the start position, the “sf_lreal” function inserts it automatically (except if start = 1). If the start value is greater than the number of characters in string, the “sf_lreal” function pads the space with blanks.
string:	Contains a string. The converted LREAL number is appended to this string at the start position. If "string" contains more characters than specified in "start", the remaining characters are truncated.
lreal:	Number of LREAL type. The number is converted to a string and appended to STRING, beginning at the start position.
length:	Specifies the length for the converted string. If length = 0 the string is formatted to the length determined by the actual number of places. If the number of required places is greater than the length value, “sf_lreal” returns the function result -1.

Parameter(s):	***Value/constant:***	***Explanation:***
format:	form_left	Format left-justified
	form_nosp	No leading blank generation
	form_sign	Always generate preceding sign: also +

The system constants consist of a bit pattern. One bit is available for each formatting option. To use several formatting options simultaneously, you can combine the system constants in an OR operation.

Example:

```
ld      format_left
or      form_nosp
st      form_strg
```

Programming example:

```
VAR
  string        STRING
  lreal         LREAL
  length        INT
  format        WORD
VAR_END

ld              20
sf_lreal        string, lreal, length, format
```

Function result:

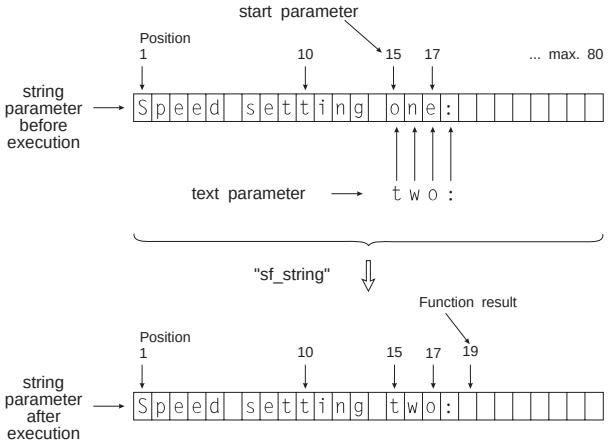
If executed successfully, the function returns the position where it finished the operation. This supplies the new start value for a subsequent string formatting function. If an error occurs, the function returns -1. This would result in a start value of less than 0 for a subsequent string formatting function. In this case, the string formatting function decrements and returns the start value.

NOTE
This is valid for all string functions.

7.2.99 sf_string, Insert or replace text

Description:

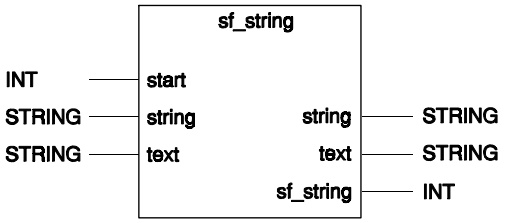
This function is used for inserting or replacing substrings. The "string" and "text" parameters contain strings. The text is appended to the string, beginning at the position specified with the start parameter. The characters are overwritten from the start position.



Block header:

```
Name:           sf_string
Type:           CONTROLLER_FUN

VAR_INPUT
  start         INT
VAR_END

VAR_OUTPUT
  sf_string     INT
VAR_END

VAR_IN_OUT
  string        STRING
  text          STRING
VAR_END
```

FBD representation



Parameter(s):	***Explanation:***
start:	Specifies the starting position in "string" where the text is to be appended. If the start value is greater than the number of characters in string, the “sf_string” function pads the space with blanks.
string:	Contains the original string.
text:	Contains the string to be appended to "string".

Programming example:

```
VAR
  string        STRING
  text          STRING
VAR_END

ld              17
sf_string       string, text
```

Function result:

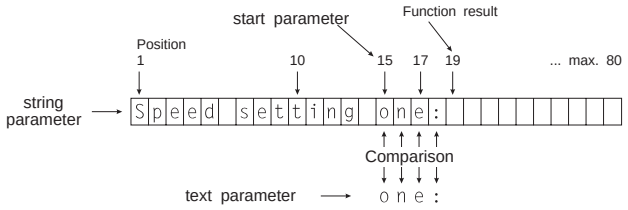
If executed successfully, the function returns the position where it finished the operation. This supplies the new start value for a subsequent string formatting function. If an error occurs, the function returns -1. If the input value (the start position) is negative, the input value is decremented and returned as the function result.

NOTE
This is valid for all string functions.

7.2.100 si_comp, Compare text

Description:

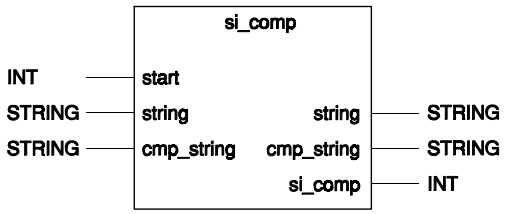
This function compares strings. The text parameter is compared to the string parameter. The comparison starts at the start position specified with the start parameter.



Block header:

```
Name:           si_comp
Type:           CONTROLLER_FUN

VAR_INPUT
  start         INT
VAR_END

VAR_OUTPUT
  si_comp       INT
VAR_END

VAR_IN_OUT
  string        STRING
  cmp_string    STRING
VAR_END
```

FBD representation



Parameter(s):	***Explanation:***
start:	Specifies the start position in the "text" string for the comparison.
string:	Contains the string to be compared with "text".
text:	Contains the string to be compared with the "string" parameter.

Programming example:

```
VAR
  string        STRING
  cmp_string    STRING
VAR_END

ld              'Command '
st              TEXT

ld              17
si_comp         string, text
```

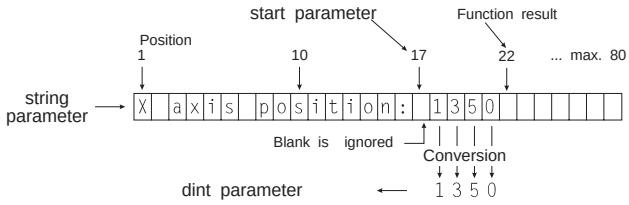
Function result:

If executed successfully, the CR contains the position at which the function finished the operation.
If an error occurs, the function returns -1.
If a negative number is specified as the start position, the “si_comp” function decrements this number and terminates. This is the case when an error occurred in a previously called string function. A suitable error handling routine can be used to determine which function generated the error.

7.2.101 si_dint, Convert number string to DINT type

Description:

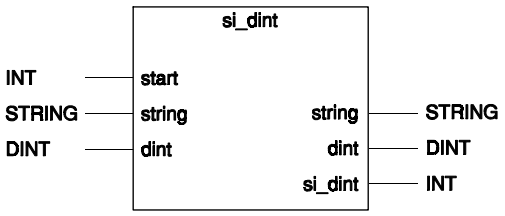
This function searches a string for a number and converts it to DINT type. Preceding signs are evaluated (2# for binary number and 16# for hexadecimal number). Leading blanks are ignored.



Block header:

```
Name:           si_dint
Type:           CONTROLLER_FUN

VAR_INPUT
  start         INT
VAR_END

VAR_OUTPUT
  si_dint       INT
VAR_END

VAR_IN_OUT
  string        STRING
  dint          DINT
VAR_END
```

FBD representation



Parameter(s): ***Explanation:***

start:	Specifies the start position from which the string is searched for a number. If a number is found, the “si_dint” function converts it to a number of DINT type. Leading blanks are ignored.
string:	Contains the string with the number.
dint:	This parameter is used for storing the converted number.

Programming example:

```
VAR
  string        STRING
  dint          DINT
VAR_END

ld              18
si_dint         string, dint
```

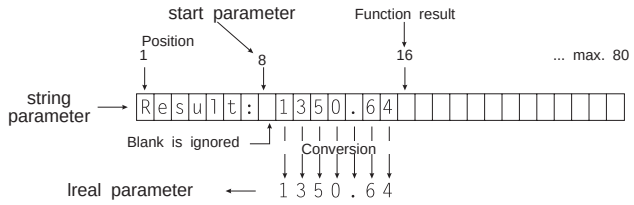
Function result:

If executed successfully, the CR contains the position at which the function finished the operation.
If an error occurs, the function returns -1.
If a negative number is specified as the start position, the “si_dint” function decrements this number and terminates. This is the case when an error occurred already in a previously called string function. A suitable error handling routine can be used to determine which function generated the error.

7.2.102 si_lreal, Convert number string to LREAL type

Description:

This function searches a string for a number and converts it to LREAL type. Preceding signs, scientific notation with “e” or “E” are evaluated. Leading blanks are ignored.

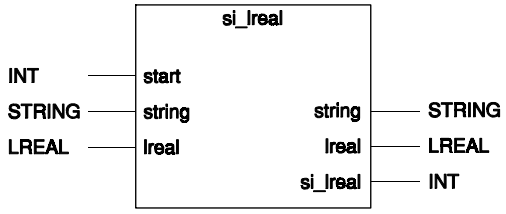


Block header:

Name: si_lreal
Type: CONTROLLER_FUN

```
VAR_INPUT
  start          INT
VAR_END

VAR_OUTPUT
  si_lreal       INT
VAR_END

VAR_IN_OUT
  string         STRING
  lreal          LREAL
VAR_END
```

FBD representation



Parameter(s):	***Explanation:***
start:	Specifies the start position from which the string is searched for a number. If a number is found, the “si_lreal” function converts it to a number of LREAL type. Leading blanks are ignored.
string:	Contains the string with the number.
lreal:	This parameter is used for storing the converted number.

Programming example:

```
VAR
  string         STRING
  lreal          LREAL
VAR_END

ld               10
si_lreal         string, lreal
```

Function result:

If executed successfully, the CR contains the position at which the function finished the operation.
If an error occurs, the function returns -1.
If a negative number is specified as the start position, the “si_lreal” function decrements this number and terminates. This is the case when an error occurred already in a previously called string function. A suitable error handling routine can be used to determine which function generated the error.

7.2.103 si_string, Extract substring

Description:

The “si_string” function extracts a substring from a string and stores the substring in another variable. The start position is specified with the start parameter. The end of the substring is determined by the following conditions:

1. If the function does not encounter an apostrophe as the first character, the characters to the next blank or to the end of the line are copied to string2.

2. If the first character is an apostrophe, the characters to the next apostrophe or to the end of the line are copied to string2.

Example with start = 1:

abc def ghe ⟹ ´abc´

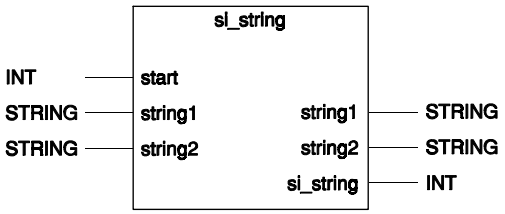
´abc def ghe ⟹ ´abc def ghe´

´abc def´ghe ⟹ ´abc def´

Block header:

```
Name:          si_string
Type:          CONTROLLER_FUN

VAR_INPUT
  start        INT
VAR_END

VAR_OUTPUT
  si_string    INT
VAR_END

VAR_IN_OUT
  string1      STRING
  string2      STRING
VAR_END
```

FBD representation



Parameter(s):	***Explanation:***
start:	Specifies the start position. Leading blanks are ignored.
string1:	Contains the string from which a substring is extracted.
string2:	This parameter is used for storing the extracted substring.

Programming example:

```
VAR
  string1      STRING
  string2      STRING
VAR_END

ld             1
si_string      string1, string2
```

Function result:

If executed successfully, the CR contains the position at which the function finished the operation. If an error occurs, the function returns -1. If a negative number is specified as the start position, the “si_string” function decrements this number and terminates. This is the case when an error occurred already in a previously called string function. A suitable error handling routine can be used to determine which function generated the error.

7.2.104 stop, Stop axis movement, controller or linear interpolation

Description:

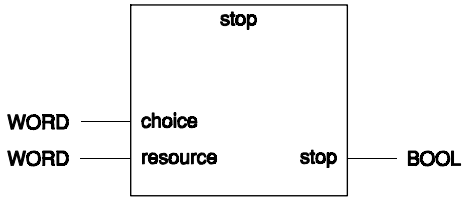
The manufacturer-defined function “stop” can be used for stopping an axis movement, the controller or a linear interpolation process. After a “stop” command, the axis is interrupted. An interrupted axis can be reenabled with the manufacturer-defined function “clrsig_sr”.

NOTE
The manufacturer-defined function “continue” can be used to resume a movement.

Block header:

```
Name:          stop
Type:          CONTROLLER_FUN

VAR_INPUT
  choice       WORD
  resource     WORD
VAR_END

VAR_OUTPUT
  stop         BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
choice:	drive	Indexer
resource:	x1, x2 ... xn	Axis 1, 2 ... n
	plc	Controller as a whole
	l1 [1)]	Linear interpolator

Programming example:

```
ld          drive
stop        x1
```

Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

1) Only available on multi-axis systems: e.g. WPM-311

7.2.105 vel, Set the set speed

Description:

The manufacturer-defined function “vel” can be used for setting the set speed of an axis. The set speed must be less than or equal to the maximum system speed (see manufacturer-defined function “setvel”). In speed mode, a movement is initiated by specifying a speed. If the value entered is greater than 0, the axis accelerates to the specified speed in positive sense of rotation. If the specified value is less than 0, it accelerates in negative sense of rotation.

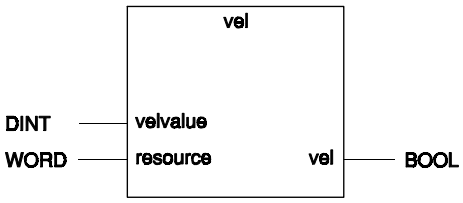
In point-to-point mode, the set speed may be changed before or during a positioning operation. In point-to-point mode, it must always be greater than 0.

ATTENTION
It is not permitted to change the speed during a reference movement or an interpolation process.

Block header:

```
Name:          vel
Type:          CONTROLLER_FUN

VAR_INPUT
  velvalue     DINT
  resource     WORD
VAR_END

VAR_OUTPUT
  vel          BOOL
VAR_END
```

FBD representation



Parameter(s):	***Value/constant:***	***Explanation:***
velvalue:	0 < Value < Max. system speed	Set speed in point-to-point mode
	- Max. s.s. < Value < + Max. s.s. (s.s. = system speed)	Set speed in speed mode
resource:	x1, x2 ... xn	Axis 1, 2 ... n

Programming example:

```
ld          10000
vel         x1
```

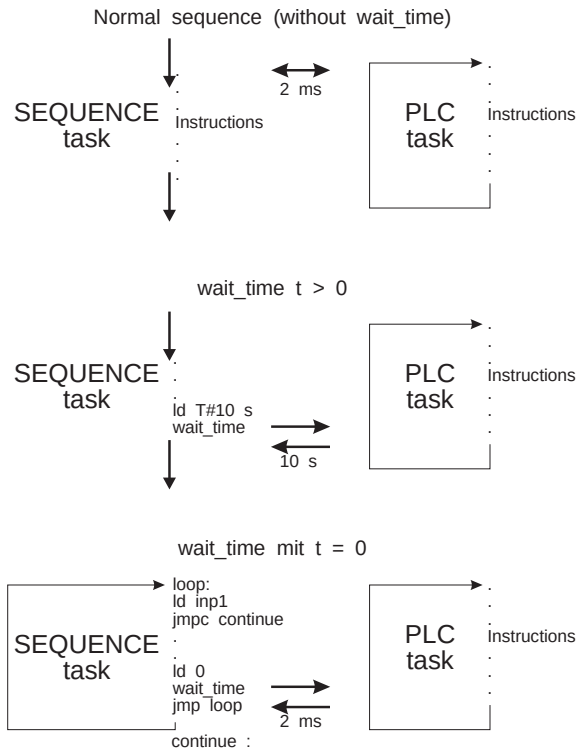
Function result:

If executed successfully, the function returns 1 (TRUE); if the function fails, it returns 0 (FALSE).
The “geterror_sr” function can be used to retrieve any error condition; see chapter 7.2.19.

7.2.106 wait_time, Wait command for SEQUENCE tasks

Description:

Usually the PLC task and the SEQUENCE task share the processing capacity, alternating every 2 ms.
The “wait_time” function can be used for specifying a wait time in the SEQUENCE program. During this period, the full processing capacity is assigned to the PLC task.
Special case: wait_time = 0.
If wait_time = 0 is specified, the PLC task is assigned the processing capacity until the next 2 ms time slice. Specifying wait_time = 0 is useful if a loop is contained in the SEQUENCE task which reads an input. For example, if the time required for processing the loop is 0.01 ms, the PLC task can utilize the remaining 1.99 ms.

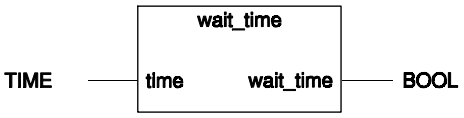


Block header:

Name: wait_time
Type: CONTROLLER_FUN

```
VAR_INPUT
  time          TIME
VAR_END

VAR_OUTPUT
  wait_time     BOOL
VAR_END
```

FBD representation



Parameter(s):	***Explanation:***
time:	Specifies the wait time. 0 waits until the next time slice is assigned.

Programming example:

```
ld              T#10 s
wait_time
```

Function result:

The function always returns TRUE.

8 Appendix

8.1 System constants

The following table lists the system constants, required for the controller library functions. The system constants are included in the assignment list of the programming system. Their data type is WORD and they comprise a defined bit pattern. The name of a system constant can be modified.

ATTENTION
The bit pattern of a system constant must not be modified.

System constant name	Meaning	Bit pattern
absolut	Absolute position of resolver	16#0005
accel	Acceleration	16#0001
accundef	Acceleration not defined	16#0004
action	Action status	16#0000
active	Axis moves (or interpolation active)	16#0002
actual	Actual value	16#1000
ad1	Analog module slot 51	16#6100
ad2	Analog module slot 53	16#6101
ad3	Encoder module with analog input in slot 55	16#6102
allsig	All signals	16#FFFF
ampdis	Power controller not yet enabled	16#0001
ampnotready	Power controller not ready	16#0400
ampoff	Power controller enable OFF	16#0004
ampon	Power controller enable ON	16#0003
ampready	Power controller ready	16#0400
amptemp	Power controller temperature prewarning	16#0800
batt	Battery	16#1200
blocked	Axis blocked	16#0008
brake	Axis decelerates	16#0004
brake1	Brake	16#0001
break_on	Activate break detection	16#0080
breaked	Axis interrupted	16#0004
c1, c2	Serial interface 1, serial interface 2	16#6300, 16#6301
catch	Frame	16#0003
com	Interface status	16#0005
command	Manipulated variable of position controller	16#0009
constant	Constant movement	16#0002
cont	Previously set value	16#4000
devnotaddressed	Device not addressed	16#0010
difference	Following error	16#0004
dirinvert	Direction inverted	16#0005
dirnormal	Direction normal	16#0006

System constant name	Meaning	Bit pattern
dragerr	Contouring error	16#0100
drive	Indexer	16#0000
encdouble	Double resolution	16#0003
encerr	Encoder error (hardware)	16#0200
encoder	Mode selection parameter	16#0001
encpulsdir	Pulse and direction	16#0005
encquad	Quadruple resolution	16#0002
encreset	Trigger reset 12024	16#0001
encsingle	Single resolution	16#0004
endcharnotfound	End-of-string character not found	16#0002
err_acc_list	Invalid master curve	16#0020
err_all	General error word	16#0000
err_break	Break level on interface	16#8000
err_calc	Value cannot be calculated	16#0800
err_charoverrun	Overrun error	16#1000
err_choice	Invalid selection	16#0400
err_com	Special interface error word	16#0001
err_comnotinit	Attempt to access non-initialized interface	16#0002
err_comoccupied	Hardware interface already assigned	16#0010
err_condition	Condition error	16#2000
err_drive_move	Command invalid while axis moving	16#0800
err_drive_stopped	Axis stopped	16#0010
err_framing	Framing error	16#4000
err_ibufoverrun	Receive buffer overflow	16#0800
err_ipo_activ	Command invalid during interpolation process	16#0200
err_mem_alloc	Error in allocating/disallocating communication memory	16#0020
err_parameter	Parameter error	16#1000
err_parity	Parity error	16#2000
err_poslist	Error in processing a position list	16#0002
err_recbufsize	Receive string buffer too small	16#0040
err_ref_activ	Command invalid during a reference movement	16#0100
err_res_ready	Resource not ready	16#8000
err_sendbufdata	Invalid string in transmit string	16#0100
err_sendbufsize	Transmit string too long	16#0080
err_src_info	Insufficient information on source	16#0040
err_undef	Undefined value	16#4000
err_w_extern	External source already active	16#0004
err_waitposactiv	Wait positioning already active	16#0008
err_watch	Monitoring signal set (active)	16#0001
errbrake	Axis decelerates at emergency deceleration ramp	16#0010

System constant name	Meaning	Bit pattern
erroractiv	Error active	16#8000
even	Even parity	16#0002
form_0	Leading zeros	16#0002
form_2	Binary formatting	16#0004
form_16	Hex formatting	16#0008
form_left	Left-justified	16#0001
form_nosp	No leading blanks	16#0010
form_sign	Always generate preceding sign	16#0020
form_std	Normal formatting	16#0000
index	Index pulse	16#2000
indexer	Normalization to drive or indexer units	16#0002
indexoff	Normal index pulse evaluation	16#0006
indexreset	Index pulse sets counter to 0	16#0007
indexstop	Index pulse as a stop signal	16#0008
indextrigger	Index pulse as a trigger signal	16#0009
jump	Positional jump to position controller	16#000B
ka	Acceleration offset of position controller	16#0005
kd	Differential component of position controller	16#0003
keyall	Lock all keys	16#0002
keyfree	Unlock all keys	16#0000
keyll	Key on lower left	16#0004
keylr	Key on lower right	16#0008
keys	Front panel keys	16#1100
keyul	Key on upper left	16#0001
keyur	Key on upper right	16#0002
kf	Speed offset of position controller	16#0004
ki	Integral component of position controller	16#0002
kill	Stop position list processing	16#0001
kp	Proportional component of position controller	16#0001
l1	Linear interpolator	16#6C00
limn	Negative hardware limit switch	16#0002
limp	Positive hardware limit switch	16#0001
m2	Interbus-S master lower slot	16#6D01
mathe	Mathematical error	16#1300
maxdragerr	Maximum occurring contouring error	16#0006
maxveldrag	Contouring error at maximum speed	16#0002
minus	Axis moves in negative direction	16#0040
minveldrag	Contouring error at minimum speed	16#0003
motion	Movement status	16#0001
motortemp	Motor temperature prewarning	16#1000

System constant name	Meaning	Bit pattern
no	No parity or no handshake	16#0001
none	No mode	16#0000
notactive	Axis at a standstill (or interpolation inactive)	16#0001
odd	Odd parity	16#0003
offset	Manipulated variable offset of position controller	16#0007
p1, p2	Encoder 1 (resolver), encoder 2	16#7000, 16#7001
plc	Entire controller	16#1000
plus	Axis moves in positive direction	16#0020
pos_drag	Position following mode	16#0003
position	Point-to-point mode, position normalizing, etc.	16#0003
posover	Position overrun occurred	16#0008
posundef	Position not yet defined	16#0002
ptp	Point-to-point mode	16#0001
pulsoff	Pulse output OFF	16#0002
pulson	Pulse output ON	16#0001
recfifoempty	Receive buffer empty	16#0001
ref	Additional hardware limit switch	16#0004
refactiv	Reference movement active	16#2000
refblocked	Reference movement blocked	16#0010
referr	Reference movement error	16#4000
refnoenable	Reference movement limit switch not enabled	16#0100
refok	Reference movement successfully completed	16#1000
refsecure	Safety distance for reference movement	16#0009
resource_on	Resource available	16#0011
rfifoemptyakt	Receive buffer empty (current)	16#0200
rts_cts	Hardware handshake (RTS/CTS)	16#0003
selected	Selected acceleration	16#8000
sendfifofull	Transmit buffer full	16#0004
servo	Position deviation detection active	16#0002
sfifoemptyakt	Transmit buffer empty (current)	16#0100
source	Reference variable normalization	16#0004
srcdiff	Electronic gear following error	16#000A
srcenc	Source is an encoder	16#0001
srcnone	No source	16#0000
srcoffset	Reference variable offset	16#0007
srcvar	Source is a variable	16#0002
stand	Standstill	16#0008
start	Start speed	16#0001
stat_crcerr	Statistics: CRC error	16#0003
stat_cycles	Statistics: Data cycles	16#0001
stat_errors	Statistics: Total number of errors	16#0002

System constant name	Meaning	Bit pattern
stat_lbwerr	Statistics: Loop backword errors	16#0004
stat_moderr	Statistics: Module errors	16#0005
stop	Hardware stop	16#0008
swlimn	Negative software limit switch	16#0040
swlimp	Positive software limit switch	16#0020
swstop	Software stop	16#0080
system	Maximum system speed	16#0002
ta	Sampling time of position controller	16#0008
target	Target or set value	16#2000
trig	Trigger signal	16#0010
trig_off	Disable trigger input of signal interface	16#0009
trig_on	Enable trigger input of signal interface	16#0010
txd_off	Deactivate RS 485 driver	16#0008
txd_on	Activate RS 485 driver	16#0007
type	Controller type	16#000A
velocity	Speed mode, speed normalization, etc.	16#0002
warning	Warning status	16#0002
watch	Monitoring signals	16#0000
watchdrive	Rotation monitoring active	16#0001
x1, x2 ... xn	Axis 1, axis 2 ... axis n	16#7800, 16#7801 ... 16#780n
xdvel	Parameter for position following controller	16#0009
xon_xoff	Software handshake (XON/XOFF)	16#0002

8.2 Glossary

Acceleration curves
The shape of acceleration curves can be freely programmed. For this purpose, a *master curve* with relative speed/acceleration value pairs must be stored in a *data block*.

Access to inputs/outputs
The inputs/outputs can be accessed directly (with "@") or indirectly via the process image (with "%"). You can access a single input/output bit (with "X"), an input/output byte (with "B") or an input/output word (with "W"). When addressing inputs/outputs, the I/O module number must be specified ("0" for I/O module on controller; additional I/O modules are in preparation).

Accumulator
A register in the controller which is used for storing intermediate results. The contents of the accumulator is also referred to as the *current result (CR)*.

Arithmetic operations
The operators "add", "sub", "mul", "div" can be used to carry out arithmetic operations between one or (for "add" and "mul") several *operands* and the CR. For the operations "add" and "mul", several operands (up to a maximum of 9) can be entered using a comma delimiter.

Assignment list (AL)
The assignment list contains the symbolic names for inputs, outputs and flags.

Assignment list editor
Each project comprises an assignment list which is created with the assignment list editor ("AL editor").

Block
A block is a (sub)entity of an application program. It consists of a *block header* and a *block body*. The following block types are defined:

Program (block)
Function block
Function
Global block
Data block

Block body
The block body contains the instructions for the control program.

Block header
The block header includes information on the block itself (such as name, type, author, etc.) as well as declarations of variables and possibly function blocks.

Block header editor
This editor is used for creating a block header.

Comment editor

The comment editor can be used to insert comments into the block header.

Compiling

The blocks are stored for editing in the programming system using a defined format (source code). When downloading a project or individual blocks, the blocks are translated (compiled) from the source code into an executable code: “object code” or “pseudo-code”.

Conditional jump operations:

The modifiers “c” and “n” control the execution of the jump instructions “jmp”, “cal” and “ret”.

Jump instructions with the “c” modifier are only executed if the value in the CR is a boolean “1”, or TRUE.

Jump instructions with the “n” modifier are only executed if the value in the CR is a boolean “0”, or FALSE.

Control program

A control program consists of several program components (blocks) which are used for organizing the program in a modular structure.

Controller configuration editor

The controller configuration editor is used for defining the hardware configuration.

Controller error memory

Runtime errors are written to the controller’s error memory and indicated in the controller’s status display. A maximum of 16 errors can be stored in the controller error memory (the first 8 and the last 8 errors which occurred).

Current result (CR)

The current result (CR) is a temporary storage element (accumulator) on the controller which is used for arithmetic and logical operations and for data transfer. In contrast to conventional programmable logic controllers, BERGER LAHR Series 300 controllers only have one accumulator for the CR. The memory size of the accumulator is automatically adjusted to the data type of the operand.

Cycle time

The time required for one PLC cycle is called the cycle time.

Data block declaration

All variables declared in data blocks are global variables. A data block describes a data structure which can contain elementary and derived data types. The first step is to define the desired data structure of a data block using the *DB type editor*. You can then declare a data block of a specific data block type using the DB editor.

Data block type

A data block type describes a data structure which contains several variable types. Several data blocks can be created from one data block type with the data block editor.

Data block type editor

In order to create a data block, you first have to use the data block type editor to create a *data block type*.

Data blocks (DB)

A data block is a special type of block. It does not contain any instructions but an arbitrary number of *variables* which can be accessed from each block (*global variables*).

Data type conversion

Before an IL operation is carried out, compatibility of the data types of CR and operator must be ensured.

Data type conversion functions

If the CR and the operator contain incompatible data types, a data type conversion function must be called before the operation is carried out (see standard library).

Data types

The data type determines the allocated memory space and the range of values of a variable. There are elementary and derived data types.

Debugging

Testing a program. To test ("debug") a program or block, the instructions must be loaded in "pseudo-code" (intermediate code).

Declaration

The term declaration refers to assigning a name and allocating memory space.

Download project

Within the context of the BPRO3 programming system, loading a project into the controller is called a project download. When "downloading" a project or individual blocks, the blocks are automatically compiled and linked (if not already done before downloading).

Drive units

Drive units are processing parameters internal to the controller; they are defined as follows:
Position-related drive units = Motor steps
Speed-related drive units = Motor steps/s/256
Acceleration-related drive units = Motor steps/s^2.

Electronic gear

In *position following mode*, a normalizing factor can be applied to the reference variable (e.g. encoder) to implement step-up or step-down gearing. This is also referred to as an electronic gear. The following applies:
Drive units = Reference variable x Normalizing factor

Encoder

Sensor for motor position detection (actual position detection).

Encoder synchronization

If an encoder is used (e.g. for rotation monitoring), it must be synchronized with the axis. This involves informing the controller about how many encoder units (increments) are equivalent to one drive unit (motor step). The following applies:
Encoder units = Drive units x Normalizing factor

Encoder programming

Each encoder input can be used for reference variable input (in position following mode) or for rotation monitoring.

Encoder unit

The encoder unit (increment) is derived from the resolution (encoder marks per revolution) and from the internal evaluation factor. The following applies:
Encoder units = Encoder marks x Internal evaluation factor (single, double, quadruple)

Error classes

Runtime errors are structured according to error classes. Error classes are distinguished by the error type and the effect on the controller.

Flag addressing

When addressing flags, a leading “%” character must be specified. A flag bit can be accessed with “X”, a flag byte with “B”, and a flag word with “W”. A flag bit is always of BOOLean data type, a flag byte of BYTE type, a flag word of WORD type.

Flags

Flags are storage elements which can be accessed from any block. The controller has a dedicated memory area for flags; this area can be specified using the programming system (controller configuration).

Following error

The “getpos” or “getposd” functions can be used for determining the difference (following error) between the reference variable and the actual position of the indexer. A buffer is provided to ensure that no pulses are lost.

Following error limit

The following error limit depends on the encoder resolution and the evaluation of the encoder signal programmed with “setmode”.

Function blocks (FB)

Function blocks have the same characteristics as global blocks, however, they have to be declared in the block headers of the calling blocks, i.e. a function block type is assigned a name and allocated memory space for variables). This allows you to use several function blocks of the same type but with different names and allocated memory within a program or block.

Functions (FUN)

Functions (such as “sin”) can be used by any block. A function can have several input variables (the first input variable is always stored in the CR), however, it returns only one result. The result is passed to the calling block via the current result (CR). The local variables of functions are reinitialized after each function call. This means that the same input will always return the same result. A function does not need to be declared and is always called with the same name.

Global blocks (GLB)

Global blocks can be called from program blocks, function blocks and other global blocks. They contain input/output variables for passing data to the calling block. Global blocks can be used for storing status conditions, i.e. the local variables of global blocks are not re-initialized each time the global block is called.

Global variables

Global variables are storage elements which can be accessed from any block.

Hardware configuration (controller configuration)

The controller configuration describes the type and the hardware components of the controller which is to execute the program. This enables the programming system to check the compatibility of program and controller.

IEC 1131 standard

Part 3 of the IEC 1131 standard (IEC 1131-3) describes:
– Program organization units (blocks)
– Control of program execution (tasks)
– Programming languages, or means of program representation (e.g. IL)
–Program data (variable declaration, data types, access paths).

IL editor

This editor is used for creating an IL program.

IL networks

A “network” is a sequence of functionally related IL instructions. An IL program can consist of several “networks”. You can freely define how many instructions are combined to form a “network”.

IL program

An IL program or an IL block consists of a sequence of text instructions (IL = instruction list).

Indexer (movement profile generator)

An electrical module or software which generates signals for controlling a motor from the acceleration, speed and travel (position) parameters.

INIT task

The INIT task is used for the customer-specific default settings of the controller (e.g. output initialization, drive presettings, etc.). Program blocks assigned to the INIT task are executed once after program start. Other tasks cannot be executed until the INIT task has been processed.

Input/output variables

Input/output variables are storage elements which are used for passing data between blocks (e.g. between program blocks and function blocks). Input/output variables must be declared in the block header of the block to be called with VAR_INPUT, VAR_OUTPUT or VAR_IN_OUT.

Inputs/outputs

The controller is provided with a certain number of inputs and outputs through which sequential operations are controlled. Input/output processing can be effected simultaneously with the execution of movements.

Instruction list (IL)

Instruction list (IL) is a textual programming language for PLC programming. IL is characterized by a sequence of instructions consisting of one *operator*, one optional *modifier* and, if required for the operator, one or more *operands*.

Interpolation

Simultaneous coordinated movement of several axes (at least two) along a straight line (linear interpolation), a circular arc (circular interpolation) or any curve (spline interpolation).

Labels

Labels (the targets for the jump instructions “jmp”, “jmpc”, “jmpn”) must be placed at the start of IL networks.

Linking

In order for a program to run on the controller, links must be created between the individual program components. This process is called “linking” and is carried out automatically during a “download”.

Local variables

Local variables are storage elements which can be accessed only from within a block. Local variables are declared in the block header with VAR.

Logical operations

The operators “and”, “or” and “xor” can be used to perform logical operations with one or more boolean operands and the CR.

Master curve

The master curve is used with the “calcaccel” function to calculate the acceleration curve for the corresponding axis. For this purpose, the maximum acceleration must be specified with a number which is to be used for storing the calculated acceleration curve (a maximum of 10 different acceleration curves can be stored).

Modifiers

A modifier “c”, “n” or “(”, which modifies the execution of the operation as described below, can be appended to certain operators.

Boolean negation:

The modifier “n” can be used in a transfer operation or a logical operation to carry out a boolean negation with the operand before performing the operation itself. Negation is only possible with the BOOL, BYTE or WORD data types.

Normalizing factors

Normalizing factors are used for:
– converting user-defined units to drive units,
– synchronizing the encoder with the axis,
– implementing step-up or step-down gearing with a reference variable in position following mode (electronic gear).

Object code

Instructions compiled into object code can be directly executed on the controller. Object code instructions are faster in execution than pseudo-code (intermediate code) instructions.

Operand

All IL instructions (except “ret”) require an operand with which the operation is performed.

Operator

The operator describes the operation which is to be carried out with the *operand* and the current result (CR).

PLC task

The PLC task is called after execution of the INIT task and is executed in cycles. When the PLC task is started, the inputs are read into the process image (input/output temporary storage). After this step, the program blocks of the PLC task are processed sequentially. Subsequently the changes in the output status conditions in the process image are output and the cycle restarts.

Parentheses:

The modifier “(” can be used to modify the usual sequence of execution for logical, arithmetic and relational operations. Operations in parentheses are carried out before the operation in front of the parentheses.

Point-to-point mode

In point-to-point mode, a positioning command is used for moving from point A to point B. Positioning can be effected with absolute values (relative to the zero point of the axis) or with incremental values (relative to the current position of the axis).

Position controller

An electrical module or software which ensures that the motor’s actual position corresponds to the motor’s set position. Actual position detection is effected by installing encoders on the motor flange of stepping motors or resolvers on the motor flange of AC motors (synchronous or asynchronous motors).

Position following mode

In position following mode, setpoints are preset through an encoder input or a variable.
The setpoint is retrieved and the axis positioned using an adjustable time base (sampling time). The setpoint can be modified with a gear ratio so that an electronic gear can be implemented.

Position lists

The “setsiglist”, “setvellist” and “setnormlist” functions can be used to activate position lists which are stored in a tabular format in data blocks.

Process image (PI)

In the process image, the inputs/outputs are temporarily stored.

Program blocks (PRG)

Program blocks are program components which can be assigned to a task. Data exchange between the program blocks of a task is effected via global variables (e.g. flags). Within a program block, function blocks, global blocks and functions can be called.

Program documentation

On completion of the project the project files can be output (in the screen display formats of the various editors) to the printer or to a documentation file (.DOC). You can specify previously which information is to be output.

Program representation

The programming languages for programmable logic controllers differ essentially in their way of program representation (textual or graphic).

Textual representation:
– Instruction List language (IL)
– Structured Text language (ST)

Graphic representation:
– Function Block Diagram language (FBD)
– Sequential Function Chart language (SFC)
– Ladder Diagram language (LD)

Pseudo-code (intermediate code)

Instructions loaded as pseudo-code into the controller can be decompiled into source code, which means they can be re-edited (this is not possible with plain object code). Pseudo-code instructions must be interpreted before being executed on the controller.

Reference movement

The “refpos” “refposw” and “refposf” functions can be used for performing reference movements. Reference movements are only possible in point-to-point mode. When executing a reference movement, a reference point is approached which is to be defined as the zero point for the system of dimensions.

Relational operations

The operators “eq”, “ne”, “gt”, “ge”, “lt” and “le” can be used for relational operations between one or more operands and the CR. After the operation, the CR has always the data type BOOL and contains any of the following values:
– “0” or FALSE, if the result is not true or
– “1” or TRUE, if the result is true.

Resolver

A special encoder for AC motors.

SEQUENCE task

The SEQUENCE task is executed simultaneously with the *PLC task* (according to the time-slice principle), however, SEQUENCE task program blocks are processed only once. It is recommended to use SEQUENCE tasks primarily for movement programming. The process image is updated by the PLC task.

Setpoint (set position)

In point-to-point mode, setpoints are preset with the “pos” or “move” functions. At the same time, a positioning operation is initiated.

Setting and resetting

The set (“s”) and reset (“r”) commands can be used for setting a variable of type BOOL (e.g. output or flag) to 1 or resetting it to 0, depending on the current result.

Signal-dependent deceleration (emergency deceleration ramp)

Certain input signals can be linked to acceleration curves for quick braking, e.g. for emergency stop or reference movement. As soon as the corresponding input signal is active, the shaft is decelerated using the linked acceleration curve.

Software configuration (task configuration)

Contains information on how and when program blocks are processed.

Speed mode

In speed mode, the “vel” function is used for setting a set speed and starting a movement. The axis will move at this speed until a different set speed is defined. The speed is specified in user-defined units, e.g. 10 (cm/s).

Speeds

Fixed speeds:
The “setvel” function can be used for defining fixed speeds such as start speed and maximum system speed.
Set speeds:
The “vel” function can be used for specifying set speeds. In speed mode, an axis movement is initiated in addition.

String functions

String functions are used for formatting and interpreting character strings. They can be transmitted, for example, through the serial interface 2.

Symbolic names

Symbolic names improve the readability of a program. In a program, variables, labels and blocks can be identified with symbolic names.

System constants

Certain system constants are required for parameter setting for controller functions (e.g. axis number = “x1”, serial interface = “c2”, encoder input 1 = “p1”). These names may not be used for any other purpose (e.g. variable, block name). The name of the system constants may be changed by the user, however, the contents cannot be changed.

Task

A task is a program organization unit similar to the organization block of a conventional programmable logic controller. It controls the way and the time of execution of program blocks. One or more program blocks can be assigned to a task. When a task is activated, the program blocks are processed in the order of their entries in the task configuration list (from top to bottom).

Task configuration editor

With the task configuration editor, you can assign one or more program blocks to a task. If several program blocks are assigned to a task, they are processed in the order of their entries in the *task configuration list* (from top to bottom).

Task configuration list

The task configuration list is a list of all task assignments defined in the project. In addition to the names of the tasks, it contains its attributes and the list of blocks assigned.

Transfer operation

The “ld” and “st” transfer operations can be used for loading values into the CR and for writing values from the CR to variables. When loading (“ld”), the CR takes the same data type as the value loaded. When storing (“st”), the data types of the CR and the operand must be compatible.

Upload project

Loading a project from the controller into the programming system is called a “project upload”.

User-defined units

User-defined units are processing parameters which can be freely defined by the user. The following applies:
Drive units = User-defined units x Normalizing factor
Positions can be specified in *drive units* or in user-defined units. *Speeds* and maximum accelerations (for calculating acceleration curves) can only be specified in user-defined units.

Variable and function block declaration

Local variables, input/output variables and function blocks must be declared in the block header of the respective block with the *block header editor* prior to their use. Variable and function block declarations always start with the string VAR... and end with the string VAR_END.

Variable initialization

When declaring a variable, it can be assigned a value (initialization). The variables are initialized during program execution.

Variables

Variables are storage elements which are used in a program for data exchange and for data storage.

8.3 Abbreviations

	%	Indirect access to the process image (PI)
	&	AND operation
	@	Direct access to (from) peripheral equipment
A	AC	Alternating current
	add	Operator for addition
	AL	Assignment list
	ASCII	American Standard Code for Information Interchange
B	B	Byte
	BNET	A special BERGER LAHR network type
C	c1, c2	Serial interface 1 and 2
	cal	Function block or global block call
	CR	Current result
D	DB	Data block
	DC	Direct current
	div	Operator for division
	Doc. no.	Documentation number
E	E	Encoder
	eq	Relational operator EQUAL TO
F	FB	Function block
	FBD	Function Block Diagram language
	FBD	Function block diagram language
	FUN	Function

G	ge	Relational operator GREATER THAN OR EQUAL TO
	gt	Relational operator GREATER THAN
H	Hz	Hertz
I	I	Input
	IEC 1131-3	Standard for PLC programming
	IL	Instruction List language
J	jmp	Jump to a label
L	ld	Load
	LD	Ladder Diagram language
	le	Relational operator LESS THAN OR EQUAL TO
	LED	Light Emitting Diode
	lt	Relational operator LESS THAN
M	M	Flag
	M	Motor
	m2	Interbus-S master in lower adapter slot
	ms	Milliseconds
	mul	Operator for multiplication
N	N	Number of encoder marks
	ne	Relational operator NOT EQUAL TO
	NWL	Network list
O	or	OR operation
P	p1, p2	Encoder 1 and 2
	PI	Process image
	PC	Personal computer
	PLC	Programmable Logic Controller
	PRG	Program block

Q	Q	Output
R	r	Reset
	Res	Result
	ret	Return from a block
S	s	Set
	SFC	Sequential Function Chart language
	st	Store
	ST	Structured Text language
	sub	Operator for subtraction
W	W	Word
X	X	Bit
	x1	Axis number
	xor	XOR operation

9 Index

S