Programming Examples
and User Library for BPRO3

BPRO3 Library

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

1 Initialization blocks

1.1 init_drive, Initialize power controller and jiggle motor

Description:

This block initializes the power controller for an axis and waits for a maximum of 1 s until the readiness signal is present. When the power controller reports “ok”, a motor jiggling movement is performed by 5 steps to the right and to the left.
Finally, the actual position of the axis is set to 0.

Can be used for these controllers:

All Series 300 units

Block header:

Name: init_drive
Type: GLB

```
VAR_INPUT
  axis          WORD      16#7800
  turn          BOOL      FALSE
  wait          BOOL      TRUE
VAR_END

VAR_OUTPUT
  ok            BOOL
  waiting       BOOL
  error         BOOL
VAR_END
```

Input parameters:

Name	Range of values	Default	Function
axis	x1, x2, x3, x4	x1	Axis identifier.
turn	TRUE, FALSE	FALSE	With turn = TRUE, jiggling is performed.
wait	TRUE, FALSE	TRUE	With wait = TRUE, the system waits for the ready signal from the power controller. With wait = FALSE, the block should be called cyclically until it reports “drive_ok” or “drive_error”. If the ready signal is not received within 1 s, the block returns “drive_error”.

Output parameters:

Name	Value	Function
ok	TRUE	Power controller has been initialized
waiting	TRUE	Wait for 1 s until ready signal is present (only for wait = FALSE)
error	TRUE	1. Message: If there is no ready signal after 1 s (only for wait = FALSE) 2. Message: If an incorrect axis number was passed

Programming example:

```
Network1 Activate power controller
  cal         init_drive(turn:=TRUE)
  ld          init_drive.error
  jmpc        disp_error
  jmpn        disp_ok

Network2 Display error in status display
  disp_error  :
  ld          99
  display
  ret

Network3 Display ok in the status display
  disp_ok     :
  ld          0
  display
  ret
```

Call the block “init_drive” to activate the power controller for axis x1. Wait until power controller returns ready and jiggling has been performed on the motor. If an error occurs during initialization, the controller status display shows “99”.

1.2 init_dragerr, Initialize rotation monitoring for axis x1

Description:

Initialize rotation monitoring for axis x1. The encoder may be connected to the encoder module p1 or p2 of the controller. Only encoders with a resolution of 500 or 1000 marks per revolution and an A/B signal interface can be used.

NOTE
Rotation monitoring must be initialized before the first positioning operation (i.e. before the reference movement).

Can be used for these controllers:

WDP5-318 and WP-311

Block header:

```
Name:   init_dragerr
Type:   GLB

VAR_INPUT
  enc           WORD      16#7000
  resolution    DINT      500
VAR_END

VAR_OUTPUT
  ok            BOOL      FALSE
VAR_END
```

Input parameters:

Name	Range of values	Default	Function
enc	p1, p2	p1	Name of encoder module
resolution	500, 1000	500	Encoder resolution, given in marks per revolution

Output parameters:

Name	Value	Function
ok	TRUE	Rotation monitoring initialized correctly
	FALSE	1. Incorrect encoder module specified 2. Incorrect resolution specified

Programming example:

```
Network1 Activate power controller
  cal       init_drive(turn:=TRUE)
  ld        init_drive.error
  jmpc      disp_error

Network2 Activate rotation monitoring
  cal       init_dragerr(resolution:=500)
  ld        init_dragerr.ok
  jmpc      disp_ok
  jmpn      disp_error

Network3 Display error in status display
  disp_error  :
  ld        99
  display
  ret

Network4 Display ok in the status display
  disp_ok   :
  ld        0
  display
  ret
```

Initialize rotation monitoring via encoder module p1 at a resolution of 500 marks per revolution.
If an error occurs during initialization, the controller status display shows “99”.

See also: init_drive

1.3 init_posdrag, Activate position following mode (electronic gear)

Description:

This block initializes the electronic gear for axis x1, with the reference variable originating from encoder module p1 or p2. The encoder module can evaluate either an A/B signal or a pulse/direction signal. The A/B signal is always subject to quadruple evaluation.
Since the electronic gear can only be initialized when the drive is at a standstill, the block aborts initialization if the axis is:

1. in point-to-point mode and moving
2. in speed mode
3. already in position following mode

The parameter force = TRUE forces the electronic gear to be initialized in any case by stopping the axis before initialization. “init_posdrag” sets the normalizing factor for the electronic gear.

Other parameters for the electronic gear must be set with the following commands (see programming manual):

	Command	**Default**
Sampling time of controller	setsrc_time	2 ms
Max. system speed	setvel	32767 Hz
Max. acceleration	accel or acclin	10 Hz/ms linear

Can be used for these controllers:

WP-311, WDP5-318, WDP3-33X

Block header:

Name: init_posdrag
Type: GLB

```
VAR_INPUT
  enc           WORD    16#7001
  mode          DINT    0
  numerator     DINT    1000
  denominator   DINT    1000
  force         BOOL    FALSE
VAR_END

VAR_OUTPUT
  ok            BOOL    FALSE
VAR_END
```

Input parameters:

Name	**Range of values**	**Default**	**Function**
enc	p1, p2	p2	Name of encoder module (p1 only for WDP5-318).
mode	0, 1	0	With mode = 0, A/B signals are subjected to quadruple evaluation. With mode = 1, pulse/direction signals are evaluated.
numerator	DINT value	1000	Gear ratio numerator.
denominator	DINT value	1000	Gear ratio denominator.
force	TRUE, FALSE	FALSE	With force = TRUE, the drive is stopped in any case before initialization. With force = FALSE, the process is aborted if the axis has **active** status.

Output parameters:

Name	**Value**	**Function**
ok	TRUE	The electronic gear has been initialized correctly
	FALSE	1. Invalid axis number 2. Invalid encoder name 3. “mode” is not 0 or 1 4. Denominator is 0 5. Axis is active (with force = FALSE)

Programming example:

```
Network1 Activate power controller
  cal           init_drive(turn:=TRUE)
  ld            init_drive.error
  jmpc          disp_error

Network2 Activate gear
  cal           init_posdrag(enc:=p2,mode:=0,numerator
                :=1000,denominator:=1000,force:=FALSE)
  ld            init_posdrag.ok
  jmpc          disp_ok
  jmpn          disp_error

Network3 Display error in the status display
  disp_error    :
  ld            99
  display
  ret

Network4 Display ok in the status display
  disp_ok       :
  ld            0
  display
  ret
```

This example initializes the electronic gear for axis x1 with encoder p2 supplying the reference variable (A/B evaluation). The gear ratio is preset with 1:1. If an error occurs during initialization, the controller status display shows “99”.

NOTE
If the gear ratio is to be changed in the current gear mode, “init_posdrag” is not useful. The “setnorm” function has been provided for this purpose (see programming manual, chapter 7.2.81).

Example:

```
ld          source
setnorm     x1, 100, 1
```

See also: init_drive

2 Movement monitoring blocks

2.1 manual, Manual movement in inching mode and continuous running

Description:

This block can be used for moving a drive via inputs and outputs. The following functions are available:
- CW and CCW rotation via two inputs
- Two speeds, selectable
- Movements with defined limit switch and stop monitoring
- Interval movements
- Application for several axes, with one axis being active at a time. (If several axes are to move simultaneously, the block can be redefined to form an FB function block; it constitutes one instance for each axis.)

In a manual movement, the frequency and the monitoring template (“ensig”) are modified according to the requirements. However, when exiting manual movement mode, all original states are restored unless any errors occurred. If possible, errors are cleared as well. As a prerequisite, the block must be called repeatedly at the end using man_enable = FALSE until it returns man_active = FALSE (however, at least once).
If a manual movement is initiated before having defined the position, the current position becomes the zero point of the system, and the position is considered to be defined.
Prerequisites for starting an axis:
- The axis must be in point-to-point mode.
- During the manual movement, no write access to this axis by any other program section is permitted.
- The absolute positioning range limit must not be exceeded.

Can be used for these controllers:

All Series 300 units

Block header:

Name: manual
Type: GLB

VAR_INPUT		
axis	WORD	16#7800
man_enable	BOOL	FALSE
left	BOOL	0
right	BOOL	0
fast	BOOL	FALSE
vel_slow	DINT	100
vel_fast	DINT	2000
tip_steps	DINT	1
snap	BOOL	TRUE
tip_delay	TIME	T#500ms
check_stop	BOOL	TRUE
VAR_END		
VAR_OUTPUT		
man_active	BOOL	
man_error	BOOL	
VAR_END		

Input parameters:

Name	Range of values	Default	Function
axis	x1, x2, x3, x4	x1	Axis identifier. If the axis identifier is changed during a movement, the manual movement of the first axis is stopped before the new one is started.
man_enable	TRUE, FALSE	FALSE	With FALSE, no movement is started and no active movement is stopped.
left	0, 1	0	Signal for CCW rotation.
right	0, 1	0	Signal for CW rotation.
fast	TRUE, FALSE	FALSE	Signal for fast or slow speed.
vel_slow		100	Slow speed.
vel_fast		2000	Fast speed.
tip_steps		1	No. of user steps executed upon each positive edge on “left” or “right”.
snap		TRUE	With snap = TRUE, a movement always ends in a defined interval pattern. With snap = FALSE, it may end at any position. The interval pattern used is always the number of user-defined steps specified with “tip_steps”.
tip_delay		T#500ms	The waiting time after moving “tip_steps” before switching over to continuous running.
check_stop	TRUE, FALSE	TRUE	With check_stop = FALSE, the STOP input is not monitored.
limit_dir	TRUE, FALSE	TRUE	With limit_dir = TRUE, only those limit switches are monitored which are located in the direction of movement.
limit_all	TRUE, FALSE	FALSE	With limit_all = TRUE, all limit switches are monitored (“limit_all” overrides “limit_dir”). When both “limit_all” and “limit_dir” are set to FALSE, no limit switches are monitored.

Output parameters:

Name	Value	Function
man_active	TRUE	A manual movement is active. If “man_enable” is set to FALSE, it is mandatory to wait until “man_active” is also set to FALSE before any other access to the same axis is allowed.
man_error	TRUE	An error occurred during a manual movement. The drive is stopped and the “swstop” flipflop is set. The user must wait until “man_active” returns FALSE (axis stopped) and then clear this flipflop using “clrsig_sr” before he can continue to move the axis.

Programming example:

```
Netzwork1   Normal manual mode
start       :
 cal        manual(man_enable:=%IX0.5,left:=%IX0.0,
            right.=%IX0.1,fast:=%IX0.8)
 ld         manual.man_active
 jmpc       error
 jmpn       start

Netzwork2   If an error occurs, wait for a axis standstill
error       :
 cal        manual(man_enable:=%IX0.5,left:=%IX0.0,
            right:=%IX0.1,fast:=%IX0.8)
 ld         manual.man_active
 jmpc       error

 ld         swstop
 clrsig_sr  x1, watch

 ld         99
 display
 jmp        start
```

In this example, the manual movement is controlled with four inputs:

IX0.0	Move to the left
IX0.1	Move to the right
IX0.5	Manual movement enable (operating mode switch)
IX0.8	Toggle fast/slow

2.2 move_plc, Relative positioning operation in PLC task

Description:

This block starts and monitors a relative positioning operation with an axis in the PLC task. The axis must be in point-to-point mode. The block takes the travel, the frequency and the ramp gradient as input parameters. The axis is started with a positive edge on the “start” input. If the axis should already be moving, it is displaced by the specified travel relative to the current position.
The frequency is also accepted without a positive edge on the “start” input. If the frequency is changed during a movement, the axis continues to run at the new frequency from this point of time.
This block can be used for moving only one axis at a time. It is necessary to wait until the axis has stopped. Only then the second axis can be started.

Can be used for these controllers:

All Series 300 units

Block header:

Name: move_plc
Type: GLB

```
VAR_INPUT
  axis          WORD    16#7800
  start         BOOL    FALSE
  distance      DINT    0
  frequency     DINT    0
  ramp          DINT    0
VAR_END

VAR_OUTPUT
  running       BOOL    FALSE
  ready         BOOL    FALSE
  error         BOOL    FALSE
VAR_END
```

Input parameters:

Name	Range of values	Default	Function
axis	x1, x2, x3, x4	x1	Axis identifier
start		0	Input or flag as a starting signal
distance		0	Travel for displacement (in user-defined units)
frequency		0	Movement speed in Hz
ramp		0	Ramp gradient for a linear ramp in Hz/ms

Output parameters:

Name	Value	Function

running	TRUE	The axis moves.
ready	TRUE	The axis reached the target.
error	TRUE	The axis is reset at the next start if: 1. the axis movement was interrupted (e.g. by limit switch) 2. an incorrect parameter was specified (incorrect axis no.) 3. the axis is not in point-to-point mode

Programming example:

```
Network1 Move axis
  cal         move_plc(axis:=x1,start:=%IX0.0,distance
              :=2500,frequency:=2000,ramp:=100)
  ld          move_plc.running
  jmpc        disp_running
  ld          move_plc.ready
  jmpc        disp_stop
  ld          move_plc.error
  jmpc        disp_error
  jmp         end

Network2 Axis stopped: Display 0
disp_stop     :
  ld          0
  display
  jmp         end

Network3 Axis running: Display 11
disp_running  :
   ld         11
  display
  jmp         end

Network4 Error: Display 99
disp_error    :
  ld          99
  display

Network5 End of block
end           :
  ret
```

In this example, the input IX0.0 starts the axis x1.
The axis states are displayed in the status display.

See also: pos_plc

2.3 pos_plc, Absolute positioning operation in PLC task

Description:

This block starts and monitors a positioning operation of an axis to a position relative to the reference point in the PLC task. The axis must be in point-to-point mode. The block takes the position, the frequency and the ramp gradient as input parameters. The axis is started with a positive edge on the “start” input. If the axis should already be moving, it is redirected to the new target position without any interruption.
The frequency is also accepted without a positive edge on the “start” input. If the frequency is changed during a movement, the axis continues to run at the new frequency from this point of time.
This block can be used for moving only one axis at a time. It is necessary to wait until the axis has stopped. Only then the second axis can be started.

Can be used for these controllers:

All Series 300 units

Block header:

Name: pos_plc
Type: GLB

VAR_INPUT		
axis	WORD	16#7800
start	BOOL	FALSE
position	DINT	0
frequency	DINT	0
ramp	DINT	0
VAR_END		
VAR_OUTPUT		
running	BOOL	FALSE
ready	BOOL	FALSE
error	BOOL	FALSE
VAR_END		

Input parameters:

Name	Range of values	Default	Function
axis	x1, x2, x3, x4	x1	Axis identifier
start		FALSE	Input or flag as a starting signal
position		0 0	Position to be approached (in user-defined units)
frequency			Movement speed in Hz
ramp		0 0	Ramp gradient for a linear ramp in Hz/ms

Output parameters:

Name	Value	Function
running	TRUE	The axis moves.
ready	TRUE	The axis reached the target.
error	TRUE	The axis is reset at the next start if: 1. the axis movement was interrupted (e.g. by limit switch) 2. an incorrect parameter was specified (incorrect axis no.) 3. the axis is not in point-to-point mode

Programming example:

```
Network1 Start axis
  cal           pos_plc(axis:=x1,start:=%IX0.0,position
                :=2500,frequency:=2000,ramp:=100)
  ld            pos_plc.running
  jmpc          disp_running
  ld            pos_plc.ready
  jmpc          disp_stop
  ld            pos_plc.error
  jmpc          disp_error
  jmp           end

Network2 Axis stopped: Display 0
disp_stop       :
  ld            0
  display
  jmp           end

Network3 Axis running: Display 11
disp_running    :
  ld            11
  display
  jmp           end

Network4 Error: Display 99
disp_error      :
  ld            99
  display

Network5 End of block
end             :
  ret
```

In this example, the input IX0.0 starts the axis x1.
The axis states are displayed in the status display.

See also: move_plc

2.4 stop_axis, Stop one or more axes; if necessary, wait until standstill

Description:

This block stops one or several axes and waits until they are at a standstill. When the axes have come to a standstill, the “clrsig_sr” command is used for resetting the error flipflop “swstop” of each axis.

Can be used for these controllers:

All Series 300 units

Block header:

Name: stop_axis
Type: GLB

```
VAR_INPUT
  stop_x1        BOOL      TRUE
  stop_x2        BOOL      FALSE
  stop_x3        BOOL      FALSE
  stop_x4        BOOL      FALSE
  wait           BOOL      TRUE
VAR_END

VAR_OUTPUT
  x1_running     BOOL
  x2_running     BOOL
  x3_running     BOOL
  x4_running     BOOL
  all_stopped    BOOL
VAR_END
```

Input parameters:

Name	Range of values	Default	Function
stop_x1	TRUE, FALSE	TRUE	Axis x1 to be stopped
stop_x2	TRUE, FALSE	FALSE	Axis x2 to be stopped
stop_x3	TRUE, FALSE	FALSE	Axis x3 to be stopped
stop_x4	TRUE, FALSE	FALSE	Axis x4 to be stopped
wait	TRUE, FALSE	TRUE	Wait until the axes are at a standstill

Output parameters:

Name	Value	Function
x1_running	TRUE	Axis x1 still running
x2_running	TRUE	Axis x2 still running
x3_running	TRUE	Axis x3 still running
x4_running	TRUE	Axis x4 still running
all_stopped	TRUE	All axes have come to a standstill.

The output parameters only refer to those axes which were stopped.

Programming example:

```
Network1 Start axis x1
  ld          1000
  vel         x1
  ld          1
  acclin      x1
  ld          5000
  move        x1

Network2 Wait for IX0.0
wait1         :
  ld          @IX0.0
  jmpn        wait1

Network3 Stop axis
wait2         :
  cal         stop_axis(stop_x1:=TRUE,wait:=FALSE)
  ld          stop_axis.x1_running
  jmpc        disp_running
  ld          stop_axis.all_stopped
  jmpc        disp_stopped
  jmp         wait2

Network4 Display 11 while axis running
disp_running  :
  ld          11
  display
  jmp         wait2

Network5 Display 0 when axis at standstill
disp_stopped  :
  ld          0
  display
  ret
```

This example starts axis x1 with a movement by 5000 steps in clockwise direction. The axis movement is stopped as soon as the input IX0.0 is energized.
The status of the axis is indicated in the status display.

2.5 wait_stand, Wait until axis is at standstill

Description:

This block waits until an axis has come to a standstill. Since this block inhibits controller enable until the axis has come to a standstill, it cannot be used in the PLC task.

Can be used for these controllers:

All Series 300 units

Block header:

```
Name:   wait_stand
Type:   GLB

VAR_INPUT
  axis          WORD        16#7800
VAR_END
```

Input parameters:

Name	Range of values	Default	Function
axis	x1, x2, x3, x4	x1	Axis identifier

Output parameters:
None

Programming example:

```
Network1 Start axis x1
  ld        2000
  vel       x1
  ld        1
  acclin    x1
  ld        10000
  move      x1

;Stop axis
  ld        drive
  stop      x1
  cal       wait_stand(axis:=x1)
  ld        swstop
  clrsig_sr x1,watch
  ret
```

See also: wait_standall

2.6 wait_standall, Wait until all axes have come to a standstill

Description:

This block waits until all axes have come to a standstill. Since this block inhibits controller enable until the axes have come to a standstill, it cannot be used in the PLC task.

Can be used for these controllers:

All Series 300 units

Block header:

```
Name:   wait_standall
Type:   GLB
```

Input parameters:
None

Output parameters:
None

Programming example:
See example on “wait_stand”

3 Blocks for the serial interface

3.1 init_com, Initialize serial interface

Description:

This block initializes the serial interface c1 or c2 with the following parameters:
– Transmit and receive buffer length
– Baud rate
– Data bits
– Stop bits
– Parity check
– Handshake
and activates the transmit line (for an RS 485 interface).
It subsequently waits for 30 ms until the connected devices are ready for reception.

Can be used for these controllers:

All Series 300 units with RS 485 or RS 232 interface

Block header:

Name: init_com
Type: GLB

```
VAR_INPUT
  port        DINT    16#6301
  fifosize    WORD    100
  baudrate    DINT    9600
  databits    BYTE    7
  stopbits    BYTE    1
  parity      BYTE    1
  handshake   BYTE    1
VAR_END

VAR_OUTPUT
  ok          BOOL
VAR_END
```

Input parameters:

Name	Range of values	Default	Function
port	c1, c2	c2	Number of serial interface
fifosize	30 ... 32767	100	Size of transmit and receive buffer in bytes
baudrate	75, 110, 150, 300, 600, 1200, 1800, 2000, 2400, 4800, 9600, 19200, 38400	9600	Baud rate
databits	6, 7, 8	7	Data bits per character
stopbits	1, 2	1	Stop bits
parity	even, odd, no	even	Parity check
handshake	no, xon_xoff, rts_cts, break_on	no	Handshake: Software handshake or hardware handshake with or without break detection

Output parameters:

Name	Value	Function
ok	TRUE	Interface correctly initialized
	FALSE	1. Incorrect interface 2. Invalid parameter

Programming example:

This example initializes interface c2 with software handshake and break detection.

```
VAR
  mode          BYTE            0
VAR_END

Netzwork1  Initialize interface c2
  ld            xon_xoff
  or            break_on
  st            mode
  cal           init_com(port:=c2,baudrate:=19200,
                databits:=8,parity:=no,
                handshake:=mode)
  ld            init_com.ok
  jmpn          disp_error
  jmpc          disp_ok

Netzwork2  Display error in status display
disp_error      :
  ld            99
  display
  ret

Netzwork3  Display ok in status display
disp_ok         :
  ld            0
  display
  ret
```

3.2 send_bus_address, Send bus address to connected unit

Description:

If one or several units are attached as slaves to the master in a bus system, the master must send the bus address before communicating with a slave. When receiving the bus address, the slave activates its transmit line.
The "send_bus_address" block sends the bus address to the connected slave controllers.

NOTE
The interface for the link must be initialized and the transmit line active.

Block header:

Name: send_bus_address
Type: GLB

```
VAR_INPUT
  port          DINT        16#6301
  address       DINT        1
VAR_END

VAR_OUTPUT
  ok            BOOL
VAR_END
```

Can be used for these controllers:

All Series 300 units

Input parameters:

Name	Range of values	Default	Function
port	c1, c2	c2	Number of the serial interface
address	1 to 31	1	Bus address

Output parameters:

Name	Value	Function
ok	TRUE	Bus address sent correctly
	FALSE	1. Incorrect interface or interface not yet initialized 2. Incorrect address specified (less than 1 or greater than 31)

Programming example:

```
Network1 Initialize interface c2
  cal           init_com
  ld            init_com.ok
  jmpn          disp_error

Network2 Send bus address 1 via c2
  cal           send_bus_address
  ld            send_bus_address.ok
  jmpn          disp_error
  jmpc          disp_ok

Network3 Display error in the status display
disp_error      :
  ld            99
  display
  ret

Network4 Display ok in the status display
disp_ok :
  ld            0
  display
  ret
```

This example initializes interface c2 for communication with a WDP5-228 controller.
The connection to the controller is also established.
Refer to the programming manual for a description of the functions and presettings used.

See also: init_com, s200_command

3.3 L_SER_CHECKQUIT, Read in and check device acknowledgement

Description:

This block checks whether an ACK message was received from a connected device. It is called by other blocks which communicate with other devices via the serial interface (e.g. “ft2_init”).

NOTE
“L_SER_CHECKQUIT” must be copied to the current project directory before using a block calling “L_SER_CHECKQUIT”.

Block header:

```
Name:  L_SER_CHECKQUIT
Type:  FB

VAR_INPUT
  com_port        WORD            16#6301
VAR_END

VAR_IN_OUT
  rx_buffer       STRING(100)     ' '
VAR_END

VAR_OUTPUT
  fb_return       INT
VAR_END
```

Can be used for these controllers:

All Series 300 units

Input parameters:

Name	Range of values	Default	Function
com_port	c1, c2	c2	Number of the serial interface

Output parameters:

Name	Value	Function
rx_buffer		Memory area in which data from the ACK message can be stored.
fb_return	1	ACK message received from external unit.
	0	No feedback received yet.
	-1	Timeout error when waiting for feedback.
	-2	NAK message received.
	-3	Error in feedback.
	-128	Other error.

3.4 s200_command, Send command to a Series 200 unit

Description:

This block can be used for addressing a Series 200 unit via the serial interface. It transmits commands to the controller and receives feedback.
The “s200_command” block also calls the library block “L_SER_CHECK_QUIT”.
This block calls another library block which is already available in the current project or must be copied previously.

Can be used for these controllers:

All Series 300 units which are attached to a WDP5-228 unit via an RS 485 bus

Block header:

Name: s200_command
Type: GLB

```
VAR_INPUT
  port          WORD          16#6301
  cmd_string    STRING(100)   ' '
VAR_END

VAR_OUTPUT
  rec_string    STRING(105)
  status        DINT
VAR_END
```

Input parameters:

Name	Range of values	Default	Function
port	c1, c2	c2	Number of the serial interface
cmd_string			Command to be transmitted

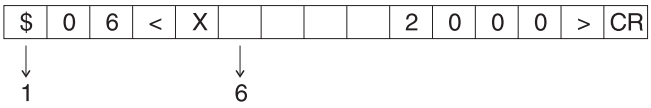
Output parameters:

Name	Value	Function
rec_string		When a status message is to be retrieved from a Series 300 unit, the message is stored in this buffer.
status	1	Feedback “ok” received from the external controller.
	-1	Timeout error when waiting for feedback.
	-2	Negative feedback received (external controller does not recognize the command).
	-3	Error in feedback (get parity error, frame error, etc. via “geterror_sr”).
	-128	Other error.

Programming example:

```
VAR
  position      DINT            0
VAR_END

Network1 Initialize interface c2
  cal           init_com

Network2 Send bus address
  cal           send_bus_address(address:=1)

Network3 Dimension setting and positioning
  cal           s200_command(cmd_string
                :='G28=0 G29E')
  cal           s200_command(cmd_string
                :='F=1000 L=50 X=2000 E')
  ld            T#5s
  wait_time

Network4 Get and evaluate position
  cal           s200_command(cmd_string:='P')
  ld            s200_command.status
  lt            0
  jmpc          disp_error

  ld            6
  si_dint       s200_command.rec_string,position
  lt            0
  jmpc          disp_error
  jmpn          disp_ok

Network5 Display error in the status display
disp_error      :
  ld            99
  display
  ret

Network6 Display ok in the status display
disp_ok         :
  ld            0
  display
  ret
```

In this example, a positioning operation is started with a WDP5-228 controller: To position 2000 at a frequency of 1000 Hz and an acceleration of 50 Hz/ms. The approached position is subsequently read. The external controller is first addressed with bus address 1 by the “send_bus_address” block via interface c2.
Reading a position is started with the ’P’ command on the WDP5-228. The feedback from the WDP5-228 is entered into the “s200_command.rec_string” parameter and has the following structure:



$06 is the ACK character (hexadecimal 06).
The actual data of the feedback are enclosed by angle brackets. The position occupies up to 8 characters next to the X; leading digit positions are padded with blanks (hexadecimal 20).
The “si_dint” command is used for interpreting the string starting at digit 6 and for storing the value in the “position” variable.

3.5 device_status, Read device status from FT 2000 or WDP5-228

Description:

This block reads the device status via the RS 485 interface. The device to be read must have been addressed (with the "send_bus_address" command).
The device status consists of a 16-bit word in which the individual bits have specific significance.

15	···	8	7	6	5	4	3	2	1	0

The assignment of the bits is as follows:

Bit	Description
0	0: Action command being processed [1] 1: No action command being processed[1]
1	0: No error active[1] 1: Error active (get error code with 'S' command) [1]
2 - 4	Not used
5	0: Output port o.k. [2] 1: Output port overload or short-circuit [2]
6	0: No fatal error [1] 1: Fatal error occurred (get error code with 'FS' command) [1]
7	0: Previous job had negative acknowledgement (not recognized) [3] 1: Previous job had positive acknowledgement [3]
8 - 15	Not used

1) WDP5-228
2) FT 2000
3) WDP5-228 and FT 2000

Can be used for these controllers:

All Series 300 units with FT 2000 or WDP5-228 connected

Block header:

Name: device_status
Type: GLB

```
VAR_INPUT
  port          WORD        16#6301
VAR_END

VAR_OUTPUT
  dev_status    WORD
  status        DINT
VAR_END
```

Input parameters:

Name	Range of values	Default	Function
port	c1, c2	c2	Number of the serial interface
wait	TRUE, FALSE	TRUE	Wait for feedback

Output parameters:

Name	Value	Function,
dev_status		See above.
status	1	Feedback received.
	-1	Timeout error when waiting for feedback.
	-2	Negative feedback received (device does not recognize the command.
	-3	Error in feedback (get parity error, frame error, etc. via geterror_sr).
	-128	Other error.

Programming example:

```
VAR
  number        DINT              0
VAR_END

Network1   Initialize interface c2
  cal           init_com(port:=c2)

Network2   Read FT 2000
  cal           send_bus_address(address:=1)
  cal           device_status
  ld            device_status.dev_status
  and           16#0020
  eq            16#0020
  st            @QX0.0

Network3   Read WDP5-228
  cal           send_bus_address(address:=2)
  cal           device_status
  ld            device_status.dev_status
  and           16#0002
  eq            16#0002
  st            @QX0.1
  retn

Network4   Display status if less than 99
  cal           s200_command(cmd_string:='S')
  ld            6
  si_dint s200_command.rec_string,number

  ld            number
  gt            0
  and(          number
  le            99
  )
  retn

  ld            number
  display
  ret
```

In the example, the device status of FT 2000 and a WDP5-228 controller is read. Both are connected in parallel to the c2 interface (RS 485). The FT 2000 has bus address 1 and the WDP5-228 controller has bus address 2.
When an overload occurs on the FT 2000 output port, the output QX0.0 is set. When the WDP5-228 controller reports an error, the output QX0.1 is set. The current status of the WDP5-228 controller is read with the 'S' command and displayed in the status display.

See also: send_bus_address, s200_command, ft2_output

4 Blocks for FT 2000

4.1 ft2_init, Initialize serial interface and address FT 2000

Description:

This block initializes a serial interface with the parameters required for transmission with the FT 2000:
– 9600 bauds
– 7 data bits
– 1 stop bit
– Even parity
– No handshake
– 100-byte transmit and receive buffer

In addition, the FT 2000 is initialized with the bus address. The bus address passed with this block must match the address set on the FT 2000. Setting the address on the FT 2000:

1. Keep the S1 key pressed while switching on.
2. Enter the password 1992<CR>.
3. Select option “1=change busaddress”.
4. Enter the bus address (address 1 is the default).

The FT 2000 must be in master/slave mode (as opposed to terminal mode).
Setting the operating mode on the FT 2000:

1. Keep the S1 key pressed while switching on.
2. Enter the password 1992<CR>.
3. Select the option “3=change mode”.
4. Enter the operating mode “1 = Master/Slave”.

This block calls another library block which is already available in the current project or must be copied previously:

L_SER_CHECK_QUIT

Can be used for these controllers:

All Series 300 units, FT 2000 version 1.04 or higher

Block header:

```
Name:   ft2_init
Type:   GLB

VAR_INPUT
  port          DINT        16#6301
  address       DINT        1
VAR_END

VAR_OUTPUT
  status        DINT
VAR_END
```

Input parameters:

Name	Range of values	Default	Function
port	c1, c2	c2	Serial interface of controller
address	1 to 31	1	Bus address

Output parameters:

Name	Value	Function
status	1	Feedback “ok” received from FT 2000.
	-1	Timeout error when waiting for feedback.
	-2	Negative feedback received (FT 2000 does not recognize the command.
	-3	Error in feedback (get parity error, frame error, etc. via “geterror_sr”).
	-128	Other error.

Programming example:

```
Network1 Initialize FT 2000
  cal           ft2_init
  ld            ft2_init.status
  lt            0
  jmpc          disp_error
  jmpn          disp_ok

Network2 Display error in the status display
disp_error      :
  ld            99
  display
  ret

Network3 Display ok in the status display
disp_ok         :
  ld            0
  display
  ret
```

This example addresses the FT 2000 with address 1 on interface c2 and waits until addressing has been completed.

4.2 ft2_fkey, Read function keys of the FT 2000

Description:

This block checks whether a function key is being pressed on the FT 2000. The FT 2000 reports whether any or no function key is pressed. It is not possible to detect whether two function keys are pressed simultaneously. If so, the FT 2000 returns the low-order key. For example, if the F3 and F4 keys are pressed simultaneously, only F3 is returned.
This block calls another library block which is already available in the current project or must be copied previously:

L_SER_CHECK_QUIT

Block header:

```
Name:   ft2_fkey
Type:   GLB

VAR_INPUT
  port          DINT        16#6301
VAR_END

VAR_OUTPUT
  fkey          DINT
  status        DINT
VAR_END
```

Can be used for these controllers:

All Series 300 units, FT 2000 version 1.04 or higher

Input parameters:

Name	Range of values	Default	Function
port	c1, c2	c2	Serial interface of controller

Output parameters:

Name	Value	Function
fkey	0 ... 8	Number of the function key which is being pressed (0 means that no function key is pressed).
status	1	Feedback received, parameter “fkey” contains the function key.
	-1	Timeout error when waiting for feedback.
	-2	Negative feedback received (FT 2000 does not recognize the command).
	-3	Error in feedback (read parity error, frame error, etc. via “geterror_sr”).
	-128	Other error.

Programming example:

```
Network1 Initialize FT 2000
  cal           ft2_init
  cal           ft2_clear

Network2 Check whether a key is pressed
wait            :
  cal           ft2_fkey

  ld            ft2_fkey.status
  lt            0
  jmpn          check1

  ld            99
  display
  jmp           wait

Network3 Display the key
check1          :
  ld            ft2_fkey.fkey
  display
  jmp           wait
```

This example checks whether a function key is being pressed. The block does not wait until a feedback is received. If a function key is pressed, its number is displayed in the controller status display.

The block checks whether
1. an error occurred
2. feedback is still awaited
3. which function key is pressed

See also: ft2_init

4.3 ft2_clear, Clear display of FT 2000

Description:

This command clears the display of the FT 2000 and moves the cursor to the home position (line 1, column 1).
This block calls another library block which must already be available in the current project directory:

L_SER_CHECK_QUIT

Can be used for these controllers:

All Series 300 units, FT 2000 version 1.04 or higher

Block header:

```
Name:   ft2_clear
Type:   GLB

VAR_INPUT
  port              WORD          16#6301
VAR_END

VAR_OUTPUT
  status            DINT
VAR_END
```

Input parameters:

Name	Range of values	Default	Function
port	c1, c2	c2	Serial interface of controller

Output parameters:

Name	Value	Function
status	1	Display cleared
	-1	Timeout error when waiting for feedback
	-2	Negative feedback received (FT 2000 does not recognize the command)
	-3	Error in feedback (get parity error, frame error, etc. via “geterror_sr”)
	-128	Other error

Programming example:

```
Network1 Initialize FT 2000
  cal           ft2_init

Network2
wait            :
  cal           ft2_clear
  ld            T#1s
  wait_time
  cal           ft2_t ext(text:='Hello World')
  ld            T#1s
  wait_time
  jmp           wait
```

See also: ft2_init, ft2_text

4.4 ft2_clrline, Clear one line

Description:

This command clears the line in which the cursor is currently located. The cursor remains at the same position.
This block calls another library block which must already be available in the current project directory:

L_SER_CHECK_QUIT

Can be used for these controllers:

All Series 300 units, FT 2000 version 1.04 or higher

Block header:

```
Name:   ft2_clrline
Type:   GLB

VAR_INPUT
  port              WORD          16#6301
VAR_END

VAR_OUTPUT
  status            DINT
VAR_END
```

Input parameters:

Name	Range of values	Default	Function
port	c1, c2	c2	Serial interface of controller

Output parameters:

Name	Value	Function
status	1	Line cleared
	-1	Timeout error when waiting for feedback
	-2	Negative feedback received (FT 2000 does not recognize the command)
	-3	Error in feedback (get parity error, frame error, etc. via “geterror_sr”)
	-128	Other error

Programming example:

This block first writes a text into both lines of the FT 2000 and then clears the lines individually.

```
Network1 Initialize FT 2000
  cal       ft2_init
  cal       ft2_clear

Network2 Output text
  cal       ft2_text(text:='Line 1 will be
            cleared now')
  cal       ft2_cursor(line:=2,col:=1)
  cal       ft2_text(text:='Line 2 will be
            cleared now')

Network3 Clear lines
  cal       ft2_cursor(line:=1,col:=10)
  ld        T#2s
  wait_time
  cal       ft2_clrline
  ld        T#2s
  wait_time
  cal       ft2_cursor(line:=2,col:=10)
  ld        T#2s
  wait_time
  cal       ft2_clrline
```

See also: ft2_init, ft2_clear

4.5 ft2_text, Output text

Description:

This block outputs a text at the current cursor position on the display.
A maximum of 40 characters can be displayed in a line. If the text is longer than the space from the cursor position to the end of the line, only the text fitting into the remaining space and the last character of the text is output. In this case, an error message is not generated.
This block calls another library block which must already be available in the current project directory:

L_SER_CHECK_QUIT

Can be used for these controllers:

All Series 300 units, FT 2000 version 1.04 or higher

Block header:

```
Name:   ft2_text
Type:   GLB

VAR_INPUT
  port              WORD            16#6301
  text              STRING(110)     ' '
VAR_END

VAR_OUTPUT
  status            DINT
VAR_END
```

Input parameters:

Name	Range of values	Default	Function
port	c1, c2	c2	Serial interface of controller
text			Text to be output

Output parameters:

Name	Value	Function
status	1	Text has been output
	-1	Timeout error when waiting for feedback
	-2	Negative feedback received (FT 2000 does not recognize the command)
	-3	Error in feedback (get parity error, frame error, etc. via “geterror_sr”)
	-128	Other error

Programming example:

```
Network1 Output a number
 cal        ft2_init
 cal        ft2_clear

 cal        ft2_text(text:='This is the number')
 cal        ft2_dint(value:=1000)

 ret
```

See also: ft2_init, ft2_clear, ft2_dint

4.6 ft2_dint, Output a DINT number on the FT 2000

Description:

This block displays a DINT number in various formats on the display. The number is output at the current cursor position. The "length" parameter can be used for specifying the length of the field which is to accommodate the number. length = 0 means that the field has the same length as the number to be output.
The "format" parameter can be used for specifying an output format such as left-justified, hexadecimal, with leading zeros. The "format" parameter is a 16-bit value, the individual bits of which have the following significance:

Bit	Bedeutung	Konstante
0	1 = Left-justified, 0 = Right-justified	form_left
1	1 = With leading zeros, 0 = without	form_0
2	1 = Binary representation [1)]	form_2
3	1 = Hexadecimal representation [1)]	form_16
4	1 = Do not generate leading blank	form_nosp
5	1 = Always generate sign	form_sign
6 - 15	Not used	

1) Bit 2 and Bit 3 = 0 represents decimal representation

This block calls another library block which must already be available in the current project directory:

L_SER_CHECK_QUIT

Can be used for these controllers:

All Series 300 units, FT 2000 version 1.04 or higher

Block header:

```
Name:   ft2_dint
Type:   GLB

VAR_INPUT
  port          WORD        16#6301
  value         DINT        0
  format        WORD        16#0000
  length        INT         0
VAR_END

VAR_OUTPUT
  status        DINT
VAR_END
```

Input parameters:

Name	Range of values	Default	Function
port	c1, c2	c2	Serial interface of controller
value		0	Number to be output
format		0 (=form_std)	Bit template for setting various output formats
length		0	Length of the field in the display to be reserved for the number

Output parameters:

Name	Value	Function
status	1	Number has been output
	-1	Timeout error when waiting for feedback
	-2	Negative feedback received (FT 2000 does not recognize the command)
	-3	Error in feedback (get parity error, frame error, etc. via “geterror_sr”)
	-128	Other error

Programming example:

The two program lines:

```
cal     ft2_text(text:='Position')
cal     ft2_dint(value:=1234,format:=0,length:=8)
```

generate the following output:

The programming example

```
VAR
  f             WORD
VAR_END

cal     ft2_text(text:='Position')
ld      form_0
or      form_nosp
st      f
cal     ft2_dint(value:=1234,format:=f,length:=8)
```

generates the following output:

See also: ft2_init, ft2_clear, ft2_text, ft2_lreal

4.7 ft2_get_dint, Get a DINT number from the FT 2000

Description:

This block waits for a DINT number to be entered on the FT 2000. A number is first displayed and can then be accepted or edited using the backspace key. The number must be confirmed by pressing the <CR> key. The "length" and "format" parameters have the same function as in the "ft2_dint" block, however, "format" does not allow selection of binary or hexadecimal representation.
This block calls another library block which must already be available in the current project directory:

L_SER_CHECK_QUIT

Can be used for these controllers:

All Series 300 units, FT 2000 version 1.04 or higher

Block header:

```
Name:   ft2_get_dint
Type:   GLB

VAR_INPUT
  port          WORD        16#6301
  default       DINT        0
  format        WORD        16#0000
  length        INT         0
VAR_END

VAR_OUTPUT
  value         DINT
  status        DINT
VAR_END
```

Input parameters:

Name	Range of values	Default	Function
port	c1, c2	c2	Serial interface of controller
default		0	Value displayed first
format		0 (=form_std)	Bit template for setting various output formats (not for "form_2" and "form_16")
length		0	Length of the field in the display to be reserved for the number

Output parameters:

Name	Value	Function
value		DINT value read in
status	1	Feedback received from FT 2000; the new number is stored in "value"
	-1	Timeout error when waiting for feedback
	-2	Negative feedback received (FT 2000 does not recognize the command)
	-3	Error in feedback (get parity error, frame error, etc. via "geterror_sr")
	-128	Other error

Programming example:

This example first outputs the number 1000 and waits until a new number is entered. The new number is then output in line 2.

```
Network1 Initialize FT 2000
  cal       ft2_init
  cal       ft2_clear

Network2 Get a number
  cal       ft2_text(text:='Enter a number')
  cal       ft2_get_dint(default:=1000)

Network3 Output the number
  cal       ft2_cursor(line:=2,col:=1)
  cal       ft2_text(text:='This was the number')
  cal       ft2_dint(value:=ft2_get_dint.value)
```

See also: ft2_init, ft2_clear, ft2_cursor, ft2_text, ft2_dint

4.8 ft2_lreal, Output an LREAL number on the FT 2000

Description:

This block displays an LREAL number in various formats on the display. The number is output at the current cursor position. The “length” and “format” parameters have the same function as in the “ft2_dint” block, however, “format” does not allow selection of binary or hexadecimal representation. This block calls another library block which must already be available in the current project directory:

L_SER_CHECK_QUIT

Can be used for these controllers:

All Series 300 units, FT 2000 version 1.04 or higher

Block header:

Name: ft2_lreal
Type: GLB

```
VAR_INPUT
  port          WORD      16#6301
  value         LREAL     0.0
  format        WORD      16#0000
  length        INT       0
VAR_END

VAR_OUTPUT
  status        DINT
VAR_END
```

Input parameters:

Name	Range of values	Default	Function
port	c1, c2	c2	Serial interface of controller
value		0.0	Number to be output
format		0 (=form_std)	Bit template for setting various output formats (not for “form_2” and “form_16”)
length		0	Length of the field in the display to be reserved for the number

Output parameters:

Name	Value	Function
status	1	Number has been output
	-1	Timeout error when waiting for feedback
	-2	Negative feedback received (FT 2000 does not recognize the command)
	-3	Error in feedback (get parity error, frame error, etc. via “geterror_sr”)
	-128	Other error

Programming example:

```
cal     ft2_init
cal     ft2_clear
cal     ft2_text(text:='This is the number')
cal     ft2_lreal(value:=3.1416,length:=10,
        format:=form_sign)
cal     ft2_cursor(line:=2,col:=1)
cal     ft2_text(text:='This is the number')
cal     ft2_dint(value:=5000,length:=10,
        format:=form_sign)
```

The following output is generated:

T	h	i	s		i	s		t	h	e		n	u	m	b	e	r		+	3	.	1	4	1	6
T	h	i	s		i	s		t	h	e		n	u	m	b	e	r				+	5	0	0	0

See also: ft2_init, ft2_clear, ft2_cursor, ft2_text, ft2_dint, ft2_get_lreal

4.9 ft2_get_lreal, Get an LREAL number from the FT 2000

Description:

This block waits for an LREAL number to be entered on the FT 2000. A number is first displayed and can then be accepted or edited using the backspace key. The number must be confirmed by pressing the <CR> key.The “length” and “format” parameters have the same function as in the “ft2_dint” block, however, “format” does not allow selection of binary or hexadecimal representation.
This block calls another library block which must already be available in the current project directory:

L_SER_CHECK_QUIT

Can be used for these controllers:

All Series 300 units, FT 2000 version 1.04 or higher

Block header:

Name: ft2_get_lreal
Type: GLB

```
VAR_INPUT
  port          WORD      16#6301
  default       LREAL     0.0
  format        WORD      16#0000
  length        INT       0
VAR_END

VAR_OUTPUT
  value         LREAL
  status        DINT
VAR_END
```

Input parameters:

Name	Range of values	Default	Function
port	c1, c2	c2	Serial interface of controller
default		0.0	Value displayed first
format		0 (=form_std)	Bit template for setting various output formats (not for “form_2” and “form_16”)
length		0	Length of the field in the display to be reserved for the number

Output parameters:

Name	Value	Function
value		LREAL value read in.
status	1	Feedback received from FT 2000; the new number is stored in “value”.
	-1	Timeout error when waiting for feedback.
	-2	Negative feedback received (FT 2000 does not recognize the command).
	-3	Error in feedback (get parity error, frame error, etc. via “geterror_sr”).
	-128	Other error.

Programming example:

```
Network1 Initialize FT 2000
start:
  cal       ft2_init (address:=1)
  cal       ft2_clear
  cal       ft2_text(text:='Enter a new number')

Network2 Read in decimal number
wait        :
  cal       ft2_get_lreal(default:=12.5)
  ld        ft2_get_lreal.status
  lt        0
  jmpc      error

Network3 Number was entered
ok          :
  cal       ft2_cursor(line:=2,col:=1)
  cal       ft2_text(text:='This was the number:')
  cal       ft2_lreal(value:=ft2_get_lreal.value)
  ld        T#3s
  wait_time
  jmp       start

Network4 Error occurred
error       :
  ld        99
  display

  ret
End of network list
```

The example waits until a decimal number has been entered.

See also: ft2_init, ft2_clear, ft2_cursor, ft2_text, ft2_dint, ft2_lreal

4.10 ft2_cursor, Cursor positioning

Description:

This block positions the cursor of the FT 2000 at a defined location. Lines are specified by 1 or 2, columns by 1 to 40. This block calls another library block which must already be available in the current project directory:

L_SER_CHECK_QUIT

Can be used for these controllers:

All Series 300 units, FT 2000 version 1.04 or higher

Block header:

Name: ft2_cursor
Type: GLB

```
VAR_INPUT
 port            WORD        16#6301
 line            DINT        1
 col             DINT        1
VAR_END

VAR_OUTPUT
 status          DINT
VAR_END
```

Input parameters:

Name	Range of values	Default	Function
port	c1, c2	c2	Serial interface of controller
line	1, 2	1	Line number
col	1 ... 40	1	Column number

NOTE
col = 4 is invalid.

Output parameters:

Name	Value	Function
status	1	Cursor has been positioned
	-1	Timeout error when waiting for feedback
	-2	Negative feedback received (FT 2000 does not recognize the command
	-3	Error in feedback (get parity error, frame error, etc. via “geterror_sr”)
	-128	Other error

Programming example:

```
Network1 Initialize FT 2000
 cal        ft2_init
 cal        ft2_clear

Network2 Prompt for input
 cal        ft2_cursor(line:=2,col:=1)
 cal        ft2_text(text:='min 1 max 10000')
 cal        ft2_cursor(line:=1,col:=1)
 cal        ft2_text(text:='Enter frequency:')

Network3 Verify the value
 cal        ft2_get_dint
 ld         ft2_get_dint.value
 lt         1
 or(        ft2_get_dint.value
 gt         10000
 )
 jmpc       error
 ret

Network4 Report error
error       :
 cal        ft2_clear
 cal        ft2_text(text:='Invalid value')
 ret
```

The program prompts for entering a movement frequency and checks it for maximum values.

See also: ft2_init, ft2_clear, ft2_cursor, ft2_text, ft2_dint, ft2_get_dint

4.11 ft2_contrast, Setting the contrast on the FT 2000

Description:

The display contrast on the FT 2000 can be adjusted in 10 steps from 0 (low contrast) to 9 (high contrast).
This block calls another library block which must already be available in the current project directory:

L_SER_CHECK_QUIT

Block header:

Name: ft2_contrast
Type: GLB

```
VAR_INPUT
 port            WORD        16#6301
 contrast        DINT        5
VAR_END

VAR_OUTPUT
 status          DINT
VAR_END
```

Can be used for these controllers:

All Series 300 units, FT 2000 version 1.04 or higher

Input parameters:

Name	Range of values	Default	Function
port	c1, c2	c2	Serial interface of controller
contrast	0 ... 9	5	Contrast level

Output parameters:

Name	Value	Function
status	1	Contrast has been set
	-1	Timeout error when waiting for feedback
	-2	Negative feedback received (FT 2000 does not recognize the command)
	-3	Error in feedback (get parity error, frame error, etc. via “geterror_sr”)
	-128	Other error

Programming example:

```
VAR
 k              DINT            0
VAR_END

Network1        Output operating instructions
 cal            ft2_init
 cal            ft2_clear
 cal            ft2_text(text:='Current contrast')
 cal            ft2_dint(value:=k)
 cal            ft2_cursor(line:=2,col:=1)
 cal            ft2_text(text:='F1 = Decrease, F2 = Increase')

Network 2       Read function keys
wait            :
 cal            ft2_fkey
 ld             ft2_fkey.fkey
 eq             1
 jmpc           down
 ld             ft2_fkey.fkey
 eq             2
 jmpc           up
 jmp            wait

Network 3       Increase contrast
up              :
 ld             k
 eq             9
 jmpc           wait
 ld             k
 add            1
 st             k
 cal            ft2_contrast(contrast:=k)
 cal            ft2_cursor(line:=1,col:=1)
 cal            ft2_text(text:='Current contrast')
 cal            ft2_dint(value:=k)
 jmp            wait

Network 4       Decrease contrast
down            :
 ld             k
 eq             0
 jmpc           wait
 ld             k
 sub            1
 st             k
 cal            ft2_contrast(contrast:=k)
 cal            ft2_cursor(line:=1,col:=1)
 cal            ft2_text(text:='Current contrast')
 cal            ft2_dint(value:=k)
 jmp            wait
```

You can use the function keys <F1> and <F2> to increase or decrease the contrast step by step.

See also: ft2_init, ft2_clear, ft2_cursor, ft2_text, ft2_dint

4.12 ft2_input, Read input port of the FT 2000

Description:

In addition to the display, the FT 2000 provides 8 digital inputs and 8 digital outputs.
This block calls another library block which must already be available in the current project directory:

L_SER_CHECK_QUIT

Can be used for these controllers:

All Series 300 units, FT 2000 version 1.04 or higher

Block header:

```
Name:   ft2_input
Type:   GLB

VAR_INPUT
 port            WORD        16#6301
VAR_END

VAR_OUTPUT
 value           BYTE
 status          DINT
VAR_END
```

Input parameters:

Name	Range of values	Default	Function
port	c1, c2	c2	Serial interface of controller

Output parameters:

Name	Value	Function
value		8-bit value with input states (bit set to 1 = input active)
status	1	Feedback received from FT 2000; the input state is stored in “value”
	-1	Timeout error when waiting for feedback
	-2	Negative feedback received (FT 2000 does not recognize the command
	-3	Error in feedback (get parity error, frame error, etc. via “geterror_sr”)
	-128	Other error

Programming example:

This example outputs the input states on the display of the FT 2000.

```
VAR
 pattern         DINT        0
 f               WORD        0
VAR_END

Network1 Initialize
 cal             ft2_init
 cal             ft2_clear
 ld              form_0
 or              form_2
 or              form_nosp
 st              f

Network2 Read in
wait             :
 cal             ft2_input
 ld              ft2_input.value
 word_to_dint
 st              pattern
 cal             ft2_cursor(line:=1,col:=10)
 cal             ft2_dint(value:=pattern,
                 length:=8,format:=f)
 jmp             wait
```

See also: ft2_init, ft2_output

4.13 ft2_output, Set outputs of FT 2000

Description:

In addition to the display, the FT 2000 provides 8 digital inputs and 8 digital outputs.
This block calls another library block which must already be available in the current project directory:

L_SER_CHECK_QUIT

Can be used for these controllers:

All Series 300 units, FT 2000 version 1.04 or higher

Block header:

```
Name:   ft2_output
Type:   GLB

VAR_INPUT
 port            WORD        16#6301
 value           BYTE        0
VAR_END

VAR_OUTPUT
 status          DINT
VAR_END
```

Input parameters:

Name	Range of values	Default	Function
port	c1, c2	c2	Serial interface of controller
value		0	Pattern from 8 bits (set bit to 1 = output **active**)

Output parameters:

Name	Value	Function
status	1	Outputs have been set
	-1	Timeout error when waiting for feedback
	-2	Negative feedback received (FT 2000 does not recognize the command)
	-3	Error in feedback (get parity error, frame error, etc. via “geterror_sr”)
	-128	Other error

Programming example:

```
Network1    Initialize FT 2000
 cal        ft2_init

Network2
 wait       :
 cal        ft2_input
 cal        ft2_output(value
            :=ft2_i nput.value)
 jmp        wait
```

The program maps all 8 inputs of the FT 2000 to its outputs.

ATTENTION
This example may only be executed if there is no dangerous mechanical equipment connected to the outputs.

See also: ft2_init, ft2_input, device_status

5 Data blocks with exponential ramps

Description:

The following data blocks contain master curves for exponential acceleration ramps for one motor type each connected to the WDP5-318 (with the D800 power amplifier).

v51117_ramp	Master curve for motor V51117 with WDP5-318
v51122_ramp	Master curve for motor V51122 with WDP5-318
v568_ramp	Master curve for motor VRDM568 with WDP5-318
v5910_ramp	Master curve for motor V5910 with WDP5-318
v5913_ramp	Master curve for motor V5913 with WDP5-318
v597_ramp	Master curve for motor V597 with WDP5-318

On the basis of a master curve, a maximum of 10 acceleration curves with a different maximum acceleration can be derived for each axis.
10 linear ramps are predefined for each axis.

Can be used for these controllers:

WDP5-318
WP-311 and WPM-311.04 with external power controller WD-008

Programming example:

In a handling system with 3 axes and 3 different motor types, the ramps are to be customized for each axis. This is done by replacing the preset linear ramps with exponential ramps in the INIT task.

```
Network1 10 Ramps for V51117 on x1
ld          10
calcaccel       x1,1,v51117_ramp
ld          20
calcaccel       x1,2,v51117_ramp
ld          30
calcaccel       x1,3,v51117_ramp
ld          50
calcaccel       x1,4,v51117_ramp
ld          75
calcaccel       x1,5,v51117_ramp
ld          100
calcaccel       x1,6,v51117_ramp
ld          150
calcaccel       x1,7,v51117_ramp
ld          200
calcaccel       x1,8,v51117_ramp
ld          300
calcaccel       x1,9,v51117_ramp
ld          400
calcaccel       x1,10,v51117_ramp
```

Data blocks with exponential ramps

Programming example:

```
;10 ramps for V51122 on x2
ld      10
calcaccel       x2,1,v51122_ramp
ld      20
calcaccel       x2,2,v51122_ramp
ld      30
calcaccel       x2,3,v51122_ramp
ld      50
calcaccel       x2,4,v51122_ramp
ld      75
calcaccel       x2,5,v51122_ramp
ld      100
calcaccel       x2,6,v51122_ramp
ld      150
calcaccel       x2,7,v51122_ramp
ld      200
calcaccel       x2,8,v51122_ramp
ld      300
calcaccel       x2,9,v51122_ramp
ld      400
calcaccel       x2,10,v51122_ramp

;10 ramps for V568 on x3
ld      20
calcaccel       x3,1,v568_ramp
ld      50
calcaccel       x3,2,v568_ramp
ld      75
calcaccel       x3,3,v568_ramp
ld      100
calcaccel       x3,4,v568_ramp
ld      150
calcaccel       x3,5,v568_ramp
ld      200
calcaccel       x3,6,v568_ramp
ld      250
calcaccel       x3,7,v568_ramp
ld      300
calcaccel       x3,8,v568_ramp
ld      400
calcaccel       x3,9,v568_ramp
ld      600
calcaccel       x3,10,v568_ramp
```

Programming example:

```
;For axis x1, select ramp no. 2 at 20 Hz/ms
ld      2
accel       x1

;For axis x2, select ramp no. 5 at 75 Hz/ms
ld      5
accel       x2

;For axis x3, select ramp no. 7 at 250 Hz/ms
ld      7
accel       x3
```

6 Sonstige Bausteine

6.1 wpm_iw_1_to_3, Eingänge der Steuerung WPM-311.004 in Merker abbilden

Beschreibung:

Die Steuerung WPM-311.004 verwaltet 20 freie Eingänge und zusätzlich 20 Endschaltereingänge (je Achse „limp“, „limn“, „ref“, „stop“, „trig“). Mit BPRO3 bis Version 3.11 ist es nur möglich, mit dem Befehl „ld“ die Eingänge IX0.0 bis IX0.14 zu lesen. Die freien Eingänge IX0.16 bis IX0.20 und die Endschalter IX0.32 bis IX0.51 können mit dem Befehl „ld“ mit BPRO3 erst ab Version 3.2 angesprochen werden.
Der Baustein „wpm_iw_1_to_3“ kopiert diese Eingänge in einen Merkerbereich und macht sie somit verfügbar. Die Merker können dann zum Beispiel mit

```
ld   %MX1000.1
```

gelesen werden. Dazu wird der Baustein „wpm_iw_1_to_3“ zyklisch aufgerufen, z.B. in der SPS-Task. Folgende Merker werden verwendet:

Freie Eingänge	%IX0.16 %IX0.17 %IX0.18 %IX0.19 %IX0.20	%MX1001.0 %MX1001.1 %MX1001.2 %MX1001.3 %MX1001.4	
Endschalter	%IX0.32 %IX0.33 %IX0.34 %IX0.35 %IX0.36 %IX0.37 %IX0.38 %IX0.39 %IX0.40 %IX0.41 %IX0.42 %IX0.43 %IX0.44 %IX0.45 %IX0.46 %IX0.47 %IX0.48 %IX0.49 %IX0.50 %IX0.51	%MX1002.0 %MX1002.1 %MX1002.2 %MX1002.3 %MX1002.4 %MX1002.5 %MX1002.6 %MX1002.7 %MX1002.8 %MX1002.9 %MX1002.10 %MX1002.11 %MX1002.12 %MX1002.13 %MX1002.14 %MX1002.15 %MX1002.16 %MX1002.17 %MX1002.18 %MX1002.19	x1 limp x1 limn x1 ref x1 stop x1 trig x2 limp x2 limn x2 ref x2 stop x2 trig x3 limp x3 limn x3 ref x3 stop x3 trig x4 limp x4 limn x4 ref x4 stop x4 trig

Bausteinkopf:

Name: wpm_iw_1_to_3
Art: GLB

Verwendbar für Steuerungen:

WPM-311.004 abhängig von BPRO3-Version (siehe oben)

Eingangsparameter:
keine

Ausgangsparameter:
keine

Programmbeispiel:

in der SPS-Task:

```
Netzwerk1
cal      wpm_iw_1_to_3
```

6.2 abs_dint, Betrag einer DINT-Zahl berechnen

Beschreibung:

Dieser Baustein berechnet den Betrag einer DINT-Zahl.

Verwendbar für Steuerungen:

Alle Geräte der Serie 300

Bausteinkopf:

```
Name:  abs_dint
Art:   FUN

VAR_INPUT
  value          DINT
VAR_END

VAR_OUTPUT
  abs_dint       DINT
VAR_END
```

Eingangsparameter:

Name	Wertebereich	Voreinstellung	Bedeutung
value			DINT-Zahl

Ausgangsparameter:

Name	Wert	Bedeutung
abs_dint		Betrag der DINT-Zahl

Programmbeispiel:

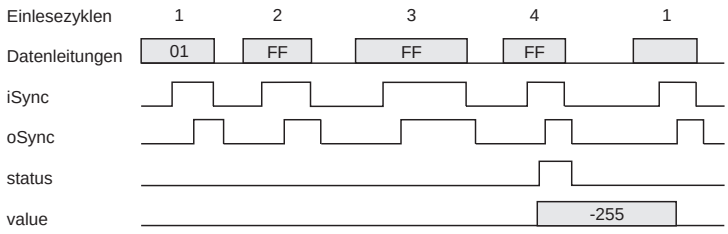
```
VAR
  position     DINT          0
VAR_END

Netzwerk1 Achse x1 starten
  ld           10000
  move         x1

Netzwerk2   Position auslesen
warten       :
  ld         actual
  getpos     x1
  abs_dint
  div        100
  st         position
  gt         99
  retc

  ld         position
  display
  jmp        warten
```

Das Beispiel liest die Position der Achse x1 und gibt sie in der Zustandsanzeige der Steuerung aus. Da der Befehl „display“ nur Werte zwischen 0 und 99 akzeptiert, muß der Positionswert vorher auf diesen Bereich abgestimmt werden. Deshalb werden nur die Hunderter- und Tausenderstellen der Position im Bereich zwischen 0 und 9999 angezeigt.

6.3 parallel8, parallele Werteingabe mit 8 Datenleitungen und 2 Steuerleitungen

Beschreibung:

Dieser Baustein liest über 8 Datenleitungen einen DINT-Wert (mit Vorzeichen) ein. Mit dem Synchronisationseingang „iSync“ meldet die externe Steuerung, daß die Daten gültig anliegen. Mit dem Synchronisationsausgang „oSync“ wird der externen Steuerung gemeldet, daß die Daten übernommen wurden. Das niederwertigste Byte muß zuerst übertragen werden.
Ein Handshake mit 4 Einlesezyklen liest jedesmal einen 32-Bit-Wert. Als Beispiel wird hier die Zahl -255 übermittelt, die hexadezimalkodiert als FFFFFF01 dargestellt wird.



Verwendbar für Steuerungen:

Alle Geräte der Serie 300

Bausteinkopf:

```
Name:   parallel8
Art:    GLB

VAR_INPUT
  iByte          BYTE      0
  iSync          BOOL      0
  cycles         DINT      4
VAR_END

VAR_OUTPUT
  value          DINT
  status         DINT
VAR_END

VAR_IN_OUT
  oSync          BOOL      0
VAR_END
```

Eingangsparameter:

Name	Wertebereich	Voreinstellung	Bedeutung
iByte			Eingangsbyte mit den 8 Datenleitungen %IB1 bedeutet Eingänge IX0.0 bis IX0.7 %IB0 bedeutet Eingänge IX0.8 bis IX0.14 (nur 7 Bit!)
iSync			Synchronisationseingang
cycles	1 bis 4	4	Anzahl der Einlesezyklen pro Wert. cycles = 4 bedeutet, daß immer 32-Bit-Werte eingelesen werden (4 x 8 Bits)
oSync			Synchronisationsausgang

ACHTUNG
Dieser Parameter darf beim Bausteinaufruf nie weggelassen werden und kann auch nur ein Prozeßabbild-Ausgang sein (%).

Ausgangsparameter:

Name	Wert	Bedeutung
value		Eingelesener Wert Je nach Anzahl der Einlesezyklen kann „value“ folgende Werte annehmen: cycles = 1 -128 bis +127 cycles = 2 -32768 bis 32767 cycles = 3 -8388608 bis 8388607 cycles = 4 -2147483648 bis +2147483647
status	1	Wert wurde eingelesen, Parameter „value“ ist gültig
	0	Einlesen läuft noch
	-1	Parameter „cycles“ nicht im Bereich 1 bis 4 Der Einlesezyklus wird abgebrochen. Die externe Steuerung muß wieder mit dem ersten Zyklus beginnen.

Programmbeispiel:

```
VAR
  flanke        r_trig
  merker        BOOL
VAR_END

Netzwerk1   FT 2000 initialisieren
  cal         ft2_init
  ld          TRUE
  st          %QX0.0

Netzwerk2   Wert einlesen
start         :
  cal         parallel8(iSync:=%IX0.8,iByte:=%IB1,
              cycles:=2,oSync:=%QX0.1)
  ld          parallel8.status
  eq          0
  st          %QX0.2

  ld          parallel8.status
  eq          1
  st          %QX0.3
  st          merker

  cal         flanke(clk:=merker)
  ld          flanke.q
  jmpn        start

Netzwerk3   Den Wert auf dem FT 2000 anzeigen
  cal         ft2_clear
  cal         ft2_text(text:='Das war der Wert')
  cal         ft2_dint(value:=parallel8.value)
  jmp         start
```

Das Programm liest über 8 Datenleitungen einen 16-Bit-Wert ein (Wertebereich -32768 bis +32767). Die Ein- und Ausgangsbelegung in dem Beispiel ist wie folgt:

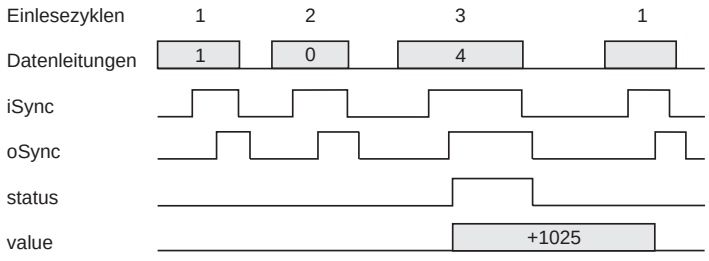
%QX0.0	Startsignal an die externe Steuerung (bereit)
%QX0.1	Synchronisationsausgang „oSync“
%QX0.2	wartet, bis Einlesen abgeschlossen
%QX0.3	Eingabe ist abgeschlossen, Wert steht in „value“
%IX0.0 bis %IX0.7	Datenleitungen
%IX0.8	Synchronisationseingang „iSync“

Siehe auch: parallel4

6.4 parallel4, parallele Werteingabe mit 4 Datenleitungen und 2 Steuerleitungen

Beschreibung:

Dieser Baustein liest über 4 Datenleitungen einen DINT-Wert (mit Vorzeichen) ein. Mit dem Synchronisationseingang „iSync“ meldet die externe Steuerung, daß die Daten gültig vorliegen. Mit dem Synchronisationsausgang „oSync“ wird der externen Steuerung gemeldet, daß die Daten übernommen wurden. Das niederwertigste Byte muß zuerst übertragen werden. Ein Handshake mit 3 Einlesezyklen liest jedesmal einen 12-Bit-Wert. Als Beispiel wird hier die Zahl +1025 übermittelt, die hexadezimalkodiert als 401 dargestellt wird.



Verwendbar für Steuerungen:

Alle Geräte der Serie 300

Bausteinkopf:

Name: parallel4
Art: GLB

```
VAR_INPUT
  iByte      BYTE    0
  iSync      BOOL    0
  cycles     DINT    8
VAR_END

VAR_OUTPUT
  value      DINT
  status     DINT
VAR_END

VAR_IN_OUT
  oSync      BOOL    0
VAR_END
```

Eingangsparameter:

Name	Wertebereich	Voreinstellung	Bedeutung
iByte			Von dem Eingangsbyte werden immer nur die unteren 4 Datenbits ausgewertet. %IB1 bedeutet Eingänge IX0.0 bis IX0.3 %IB0 bedeutet Eingänge IX0.8 bis IX0.11
iSync			Synchronisationseingang
cycles	1 bis 8	8	Anzahl der Einlesezyklen pro Wert cycles = 8 bedeutet es werden immer 32-Bit-Werte eingelesen (8 x 4 Bits)
oSync			Synchronisationsausgang

ACHTUNG
Dieser Parameter darf beim Baustein-aufruf nie weggelassen werden und kann auch nur ein Prozeßabbild-Ausgang sein (%).

Ausgangsparameter:

Name	Wert	Bedeutung
value		Eingelesener Wert Je nach Anzahl der Einlesezyklen kann „value“ folgende Werte annehmen: cycles = 1 -7 bis +8 cycles = 2 -128 bis +127 cycles = 3 -2048 bis +2047 cycles = 4 -32768 bis +32767 cycles = 5 -524288 bis +524287 cycles = 6 -8388608 bis + 8388607 cycles = 7 -134217728 bis +134217727 cycles = 8 -2147483648 bis +2147483647
status	1	Wert wurde eingelesen, Parameter „value“ ist gültig
	0	Einlesen läuft noch.
	-1	Parameter „cycles“ nicht im Bereich 1 bis 8 Der Einlesezyklus wird abgebrochen. Die externe Steuerung muß wieder mit dem ersten Zyklus beginnen.

Programmbeispiel:

Siehe: parallel8