DCS800

DriveWindow Light DWL AP

Help for DCS800 Adaptive Program Tool

Table of Contents

ABB DCS800 DriveWindow Light AP tool

Function

The DCS800 DWL AP tool creates Adaptive Program(s) with 16 function blocks as maximum. An other way for Adaptive Programming is with the DCS800 panel on the drive or parameter setting with DriveWindow Light. There are logical and arithmetical function blocks for programming inside. It is possible to connect digital in- and outputs, analog in- and outputs as well as parameters from the drive.

Buttons

The tool will be controlled with the following buttons:

Ctrl + left mouse button
→ connection of in- and outputs

Shift + left mouse button
→ view actual values

Cancel
→ abort the action

Help
→ open the Help Desktop

File

Open

Open a saved DWL AP program from drive files for editing or download. The file extension is *AP*.
It is necessary to change in Offline-Mode for opening a saved Adaptive Program.

Save DWL AP program

This command saves the Adaptive Program on the screen to the disk file. The name of this file is displayed at the top of the screen.
Note:
The save command stores all parameters of group 83 up to 86!
This is necessary for data backup and duplication.

Save As

Using this command you can save the Adaptive Program on the screen to a disk file. The extension of all DWL AP files must be *AP*.
Note:
This command saves all parameters from group 83 until 86!

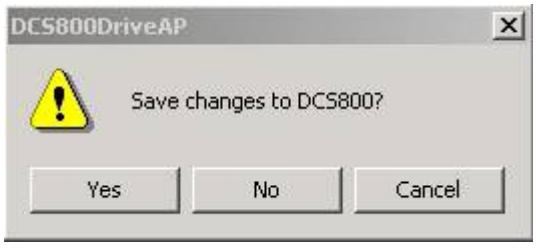
Print

Print the desktop with all the parameters and function blocks.
Choose a printer.

Exit

This command stops the execution of DWL AP.
If you close DWL AP in Online-Mode, the parameters are saved in the Drive. But the program is not saved at the PC.

Attention:
If you worked in Offline Mode and close the program, the parameters are not saved automatically.
In this case save the changes at the PC!

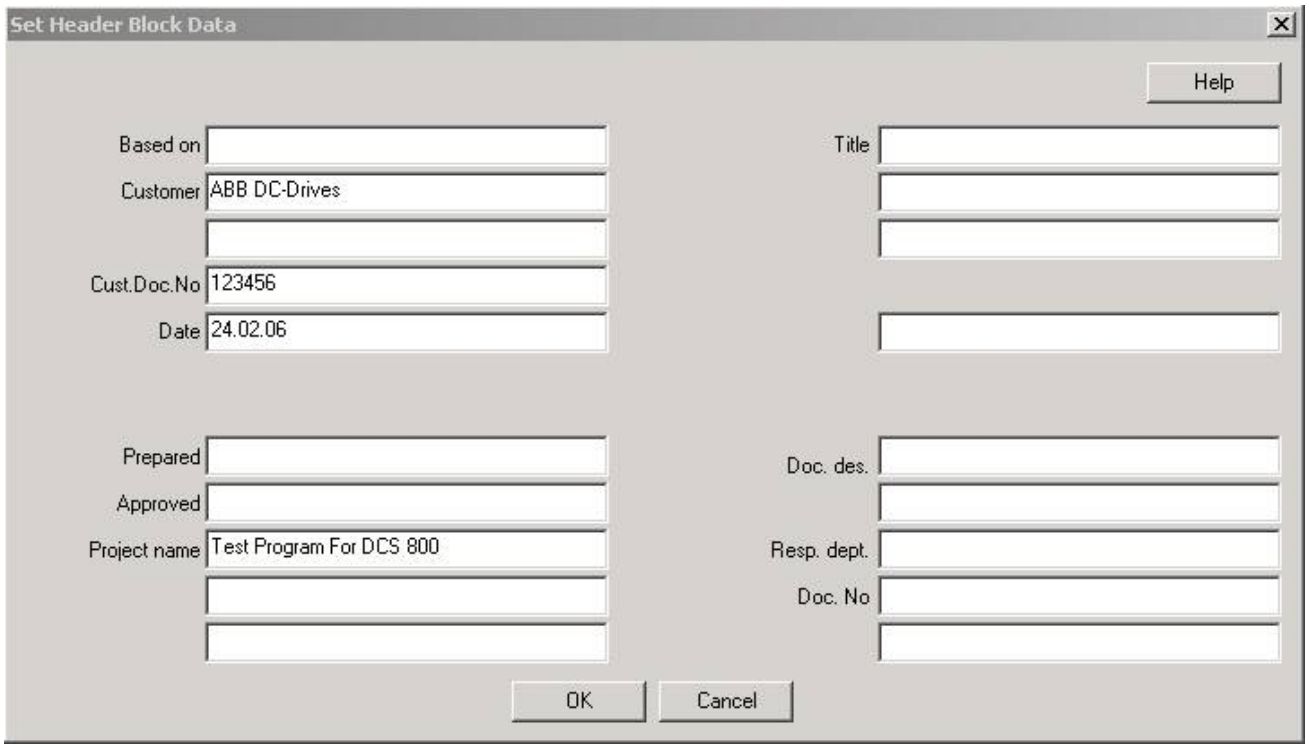


Header Block Data

For documentation use the header below the function blocks.

Based on	Prepared	Title	Doc. des.
Customer	Approved		
	Project name		Resp. dept.
Cust.Doc.No			Doc. No
Date			

Click Ctrl + left mouse button to open the window for description.



Type in important data for the created program.

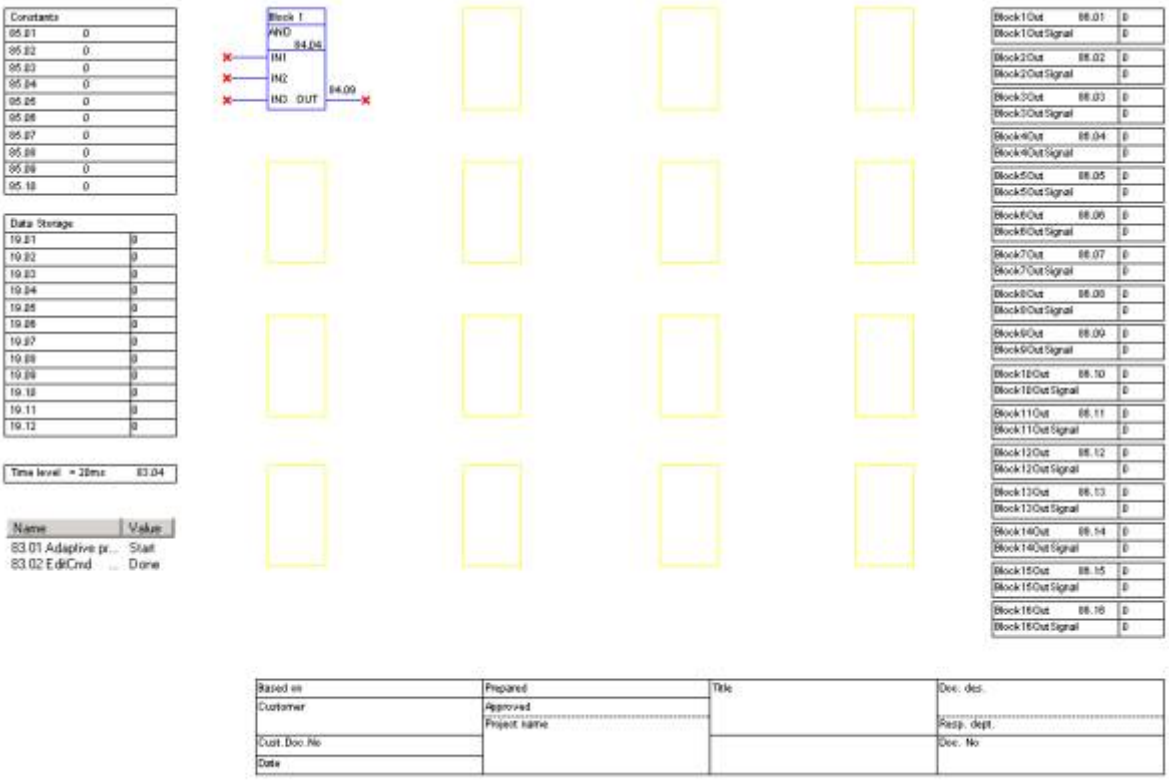
Click OK and the texts will be shown on the desktop!

Based on		Prepared	
Customer	ABB DC-Drives	Approved	
		Project name	Test Program For DCS
Cust.Doc.No	123456		
Date	24.02.06		

Edit

Copy As Bitmap

This command copies the screen's content to the clipboard (as bitmap).
Bitmap files are used by e.g. *MS Office* products. You can e.g. insert the copied picture to a WORD document by the command *Edit - Paste*.



Drive

Upload DWL AP program

If you work in *Offline Mode* it is necessary to upload the actual parameters from the drive. Then you can work in *Online Mode*.

Download DWL AP program

If you work in Offline Mode or open a saved DWL AP program it is necessary to download the parameters to the drive before starting the program.

Online

Working in *Online-Mode* is only possible if there is a active connection with a drive. Then all steps are accepted by the drive.
In this mode, all changes are modified in the drive directly.

Offline

Working in *Offline-Mode* is necessary if there is no drive available. It is possible to create programs and set parameters in *Offline-Mode*. If a drive is connected it is obligatory to download the parameters.

Start Adaptive Program

With *Start-Mode* the Adaptive Program is running. If there are faults in the program you can see them in the window above the function blocks.

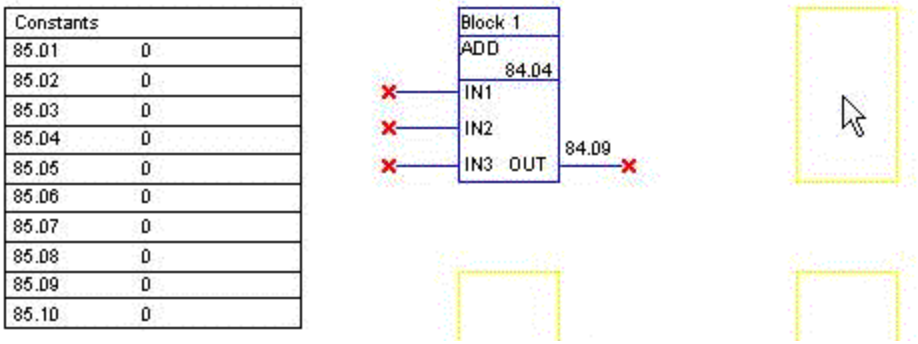
Stop Adaptive Program

The *Stop-Mode* stops the cyclic calculation of the Adaptive Program. In this mode it is not possible to change parameters. For changing the program Adaptive Program must be set to *Edit-Mode*.

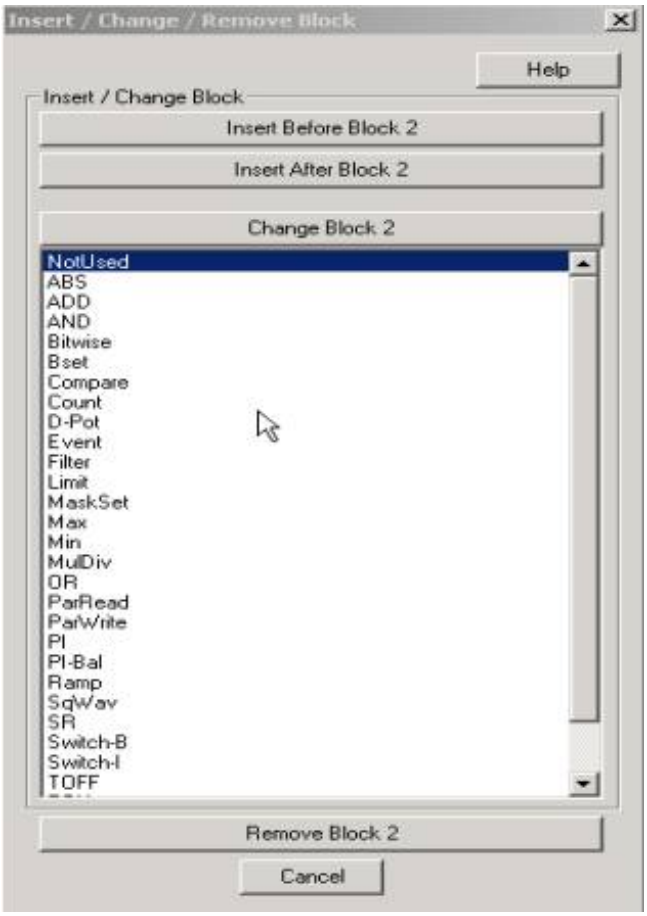
Edit Adaptive Program

The *Edit-Mode* is for creating or changing a program.

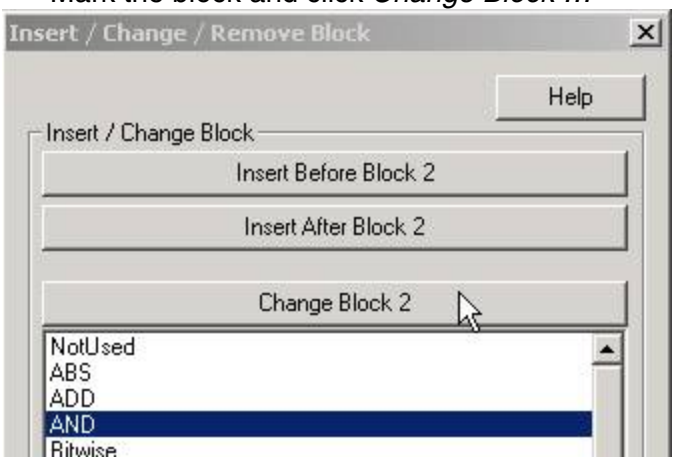
1. **Insert a function block.**
   Press Ctrl + left mouse button to the selected box.



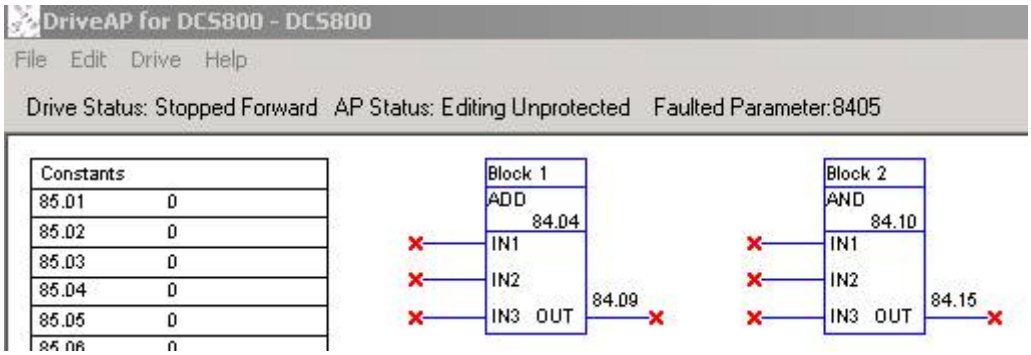
2. **A window is opened.**
   The window shows the available function blocks.



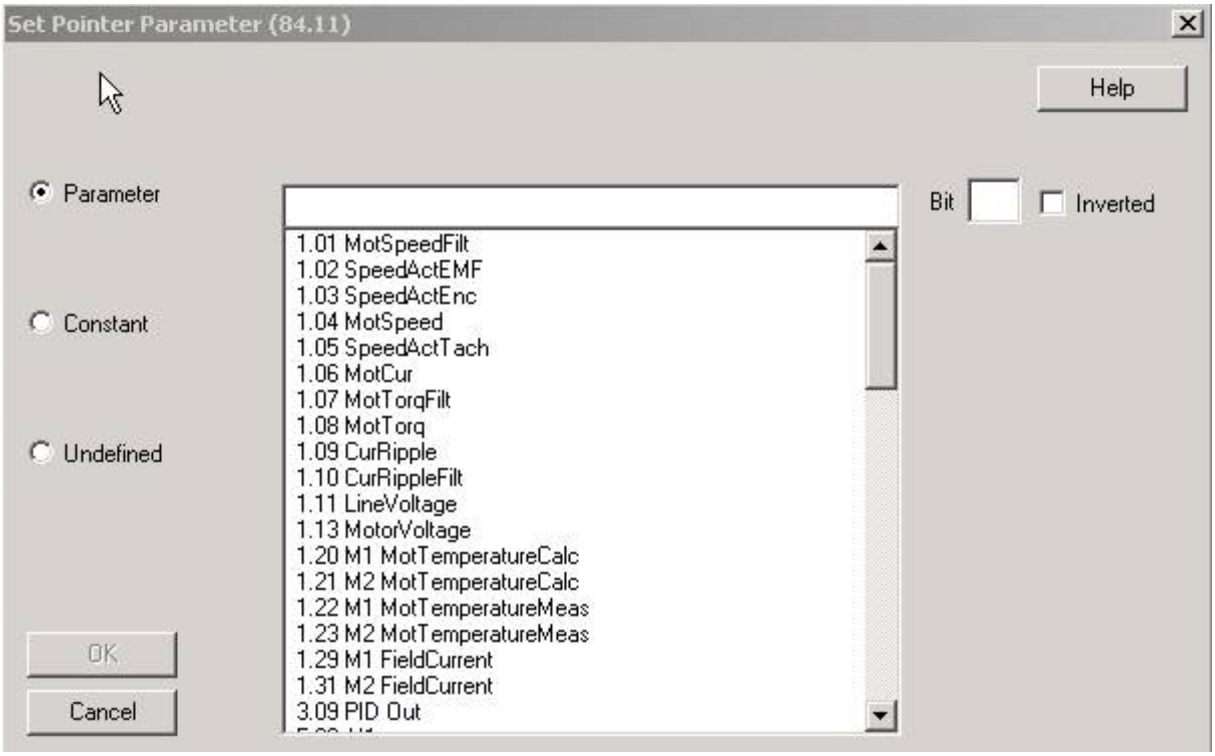
3. **Choose a function block from the list.**
   Mark the block and click *Change Block …*



4. **Now the function block is inserted.**
   You can see the new function block on the desktop.



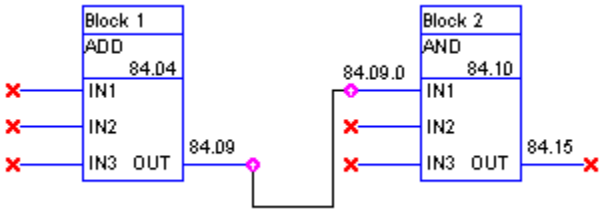
5. **Connect function blocks.**
   Select an input of a function block.
   Click the red cross with Ctrl + left mouse button.
   Now a window is opening.



Note:
It is not possible to click on an output of a function block because output pointers are displayed at the right side of the desktop.

6. Select a parameter from the list or give a constant.

Mark the parameter select the bit and click OK.



Now the blocks are connected.
For further information read *Edit AP-blocks* please.

Single Cycle

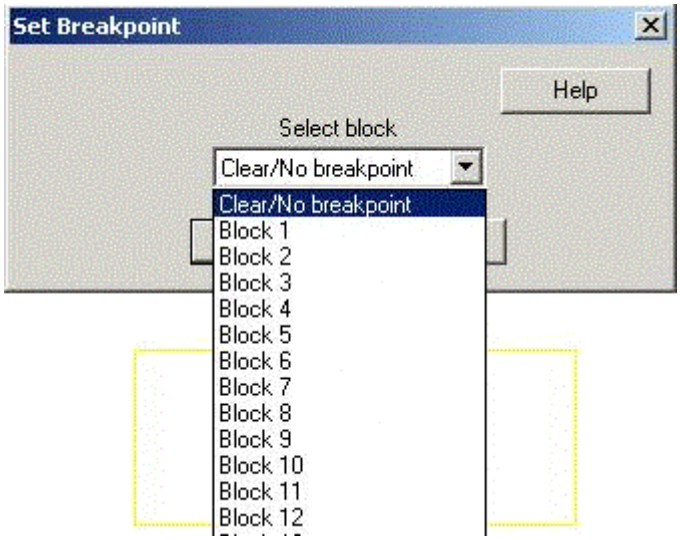
In *Single Cycle-Mode* the Adaptive Program runs only one cycle.

Single Step

In *Single Step-Mode* the tool runs only one block.

Set Breakpoint

A breakpoint is for testing the Adaptive Program. If a breakpoint is set, the tool runs to this point. Select a block for the breakpoint.

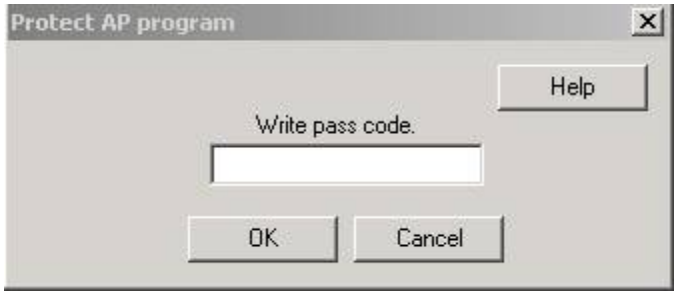


Click OK.

After this choose *Single Cycle-Mode* and the program runs up to the selected block. The selected block have a red border.
Attention:
Don´t forget to clear the breakpoint after the test.

Protect Adaptive Program

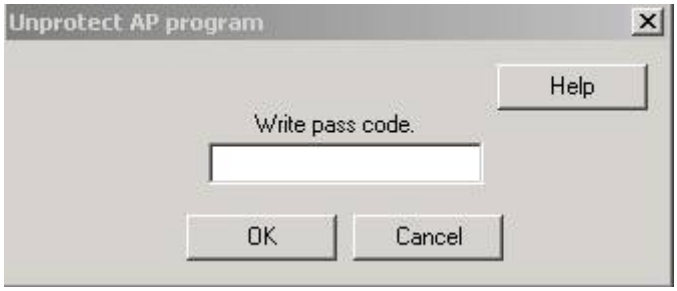
Your application can be write-protected with the help of a pass code. With setup a pass code the Adaptive Program can be protected from *Edit-Mode*.
Type in the pass code and click OK.



Now the parameter group 84 is write protected.

Unprotect Adaptive Program

For change to *Edit-Mode* unprotect the parameter group 84.
Type in the pass code and the program is unprotected.



Delete Adaptive Program

With this button the whole Adaptive Program will be deleted. In this case the parameter groups 83 up to 86 will be set to default values.

Attention:
After this the program is irreparable deleted!

Edit AP Blocks

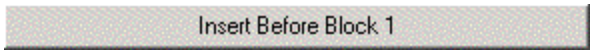
Edit AP Block

Editing of function blocks is only possible in *Edit-Mode*.
Click Ctrl + left mouse on an AP-Block to open a window.

Insert AP Block

Select a function block and mark it.

→ If you have selected the correct box.

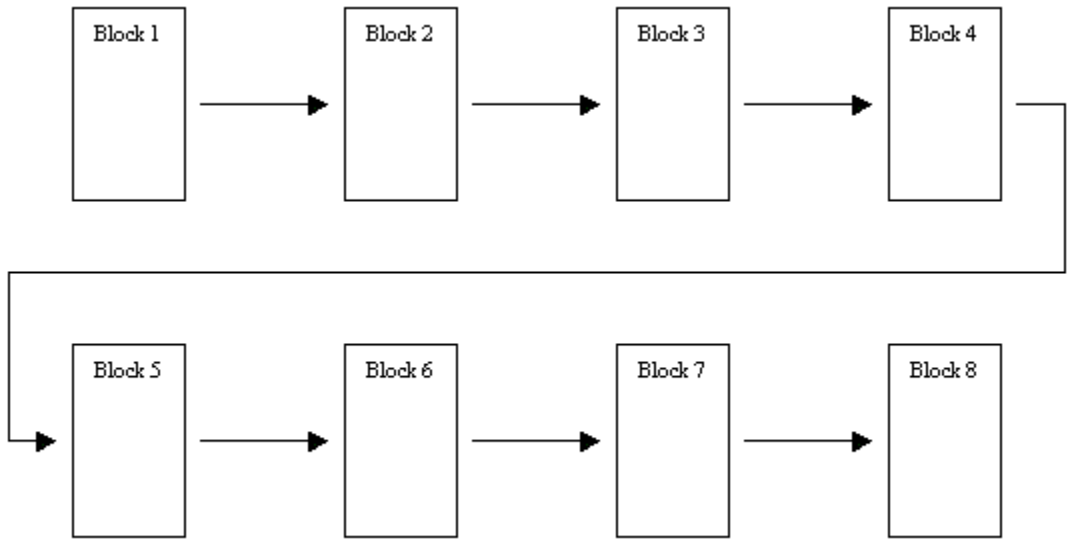


→ If you be able to insert a box before.

→ If you be able to insert a box after.

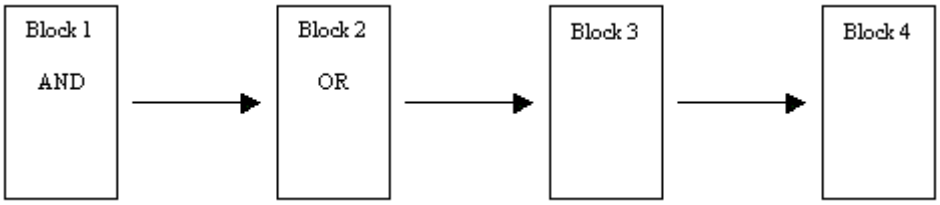
Now the block will be insert.

Attention:
The position is according to the following figure:

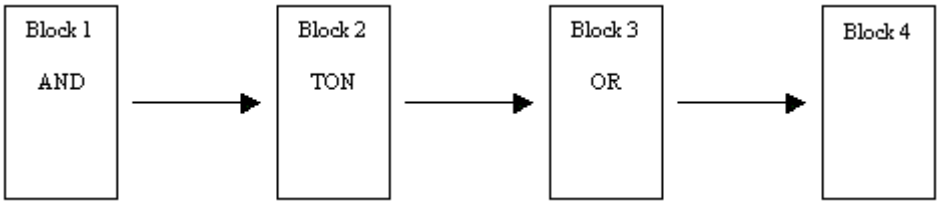


If you insert a function block before other blocks, the existing blocks will be shifted right.

Example:
Box 1 is a AND block and box 2 a OR gate. Now between box 1 and 2 a TON block should be insert.

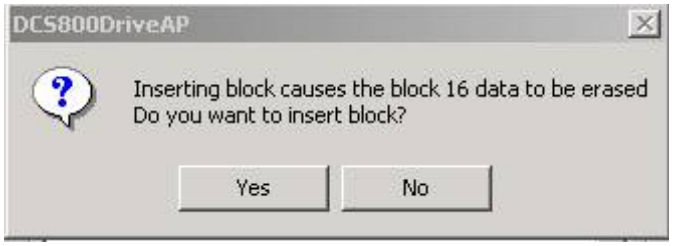


Select box 2 with Ctrl + left mouse button, choose the type TON and click *Insert Before Block 2*.



Now function block 2 will be shifted right and the function block TON will be inserted.

If all 16 boxes are reserved and a new block should be inserted, you see the message.



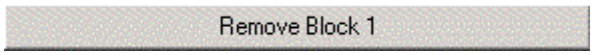
If you click *YES* block 16 is erased!

Insert after AP Block

If you select a box and would like to insert a function block after this box. Then mark the function block and click *Insert After Block…*

Now the block will be insert.

Remove AP Block



→ Press this button to delete a function block.

Attention:
If you remove a block it is irreparable deleted!

Edit AP block Type

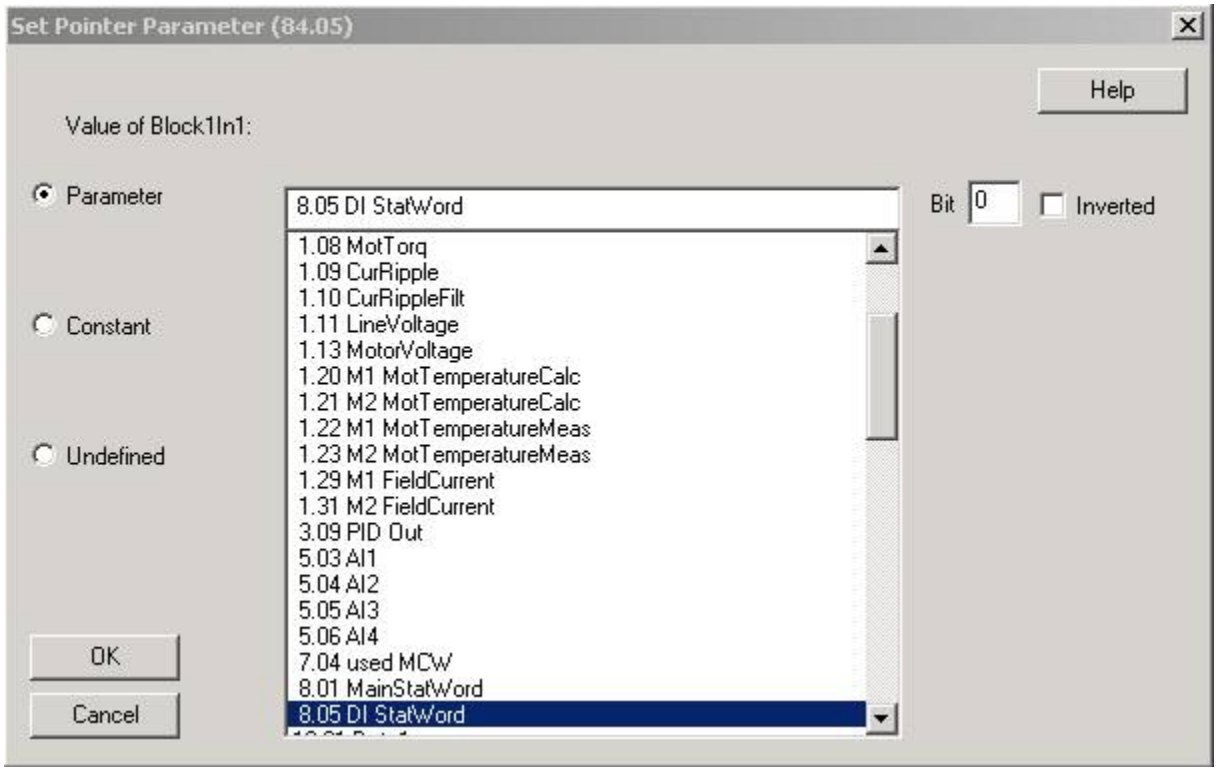
There are logical and arithmetical function blocks available.
It is possible to connect logical function blocks with other logical blocks. In the same way it is possible to connect arithmetic blocks.

Attention:
Connections of logical blocks with arithmetic blocks must be avoided!

The description of several function blocks is found at *Block Description.*

Edit input parameters

Click to an input (red cross) with the left mouse button + Ctrl.
A new window is opened:



You have the possibility to select one out of 3 input values:

Parameter

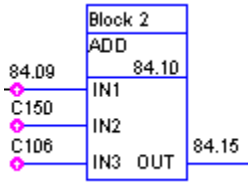
You can select from the list a parameter and the necessary bit. With the box *Inverted* the selected bit will be inverted.
The parameters are described in the parameter list for DCS 800.

Example:
Digital input 1 is selected via parameter 8.05 with bit number 0.

Constant

It is possible to set a constant value to an input.
Type in the value for the input:

At the function block you can see a **C** with the value e.g. 106.



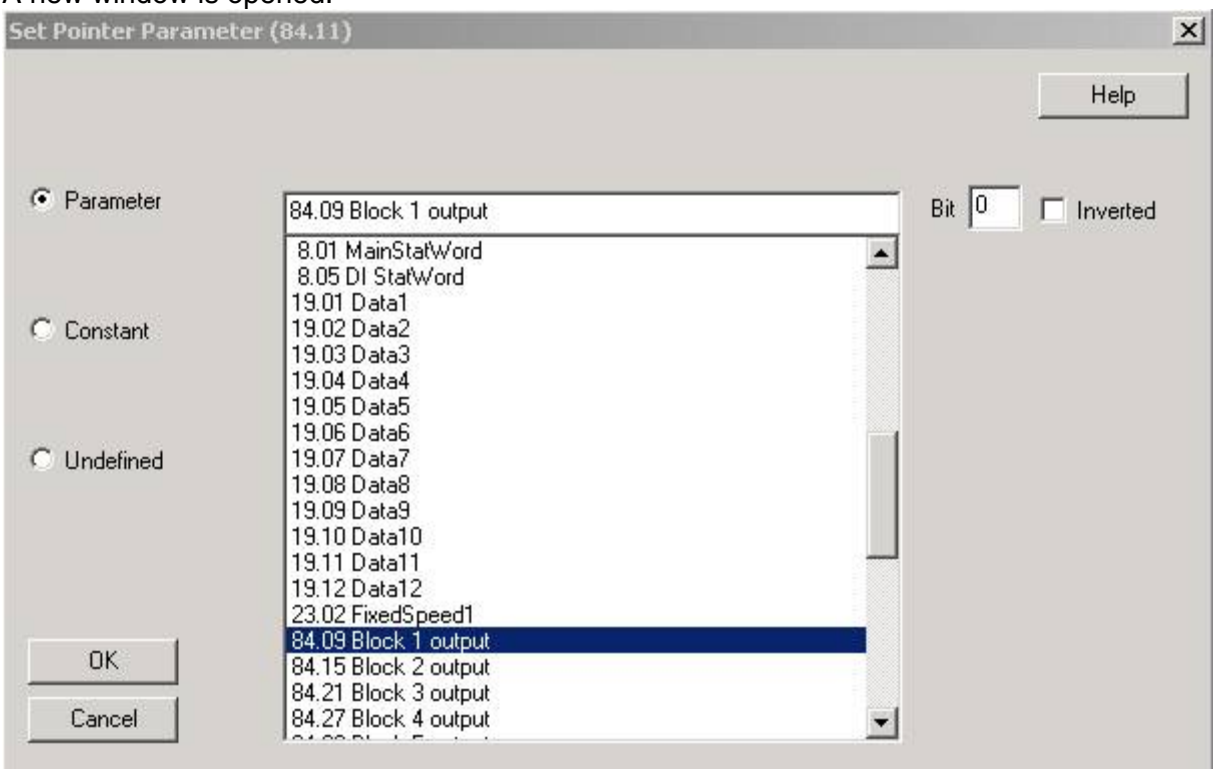
Undefined

For disconnection choose *Undefined*. In case of logic function the input is not in operation.

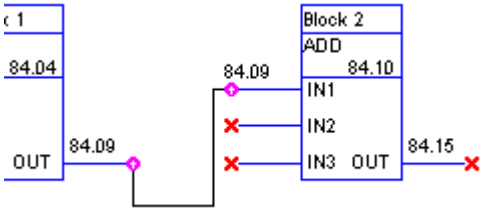
Edit input parameters (output to input)

It is possible to connect an output of a function block to an input of a block.
In this case, click to the input, that will be connected with Ctrl + left mouse button.

A new window is opened.



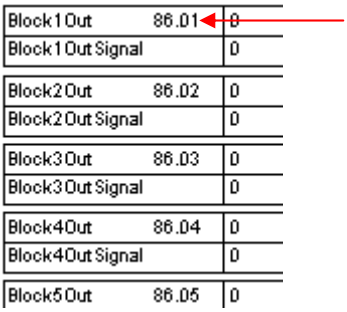
Select an output parameter of a function block in parameter menu and click OK.
Then you can see the new connection on the desktop.



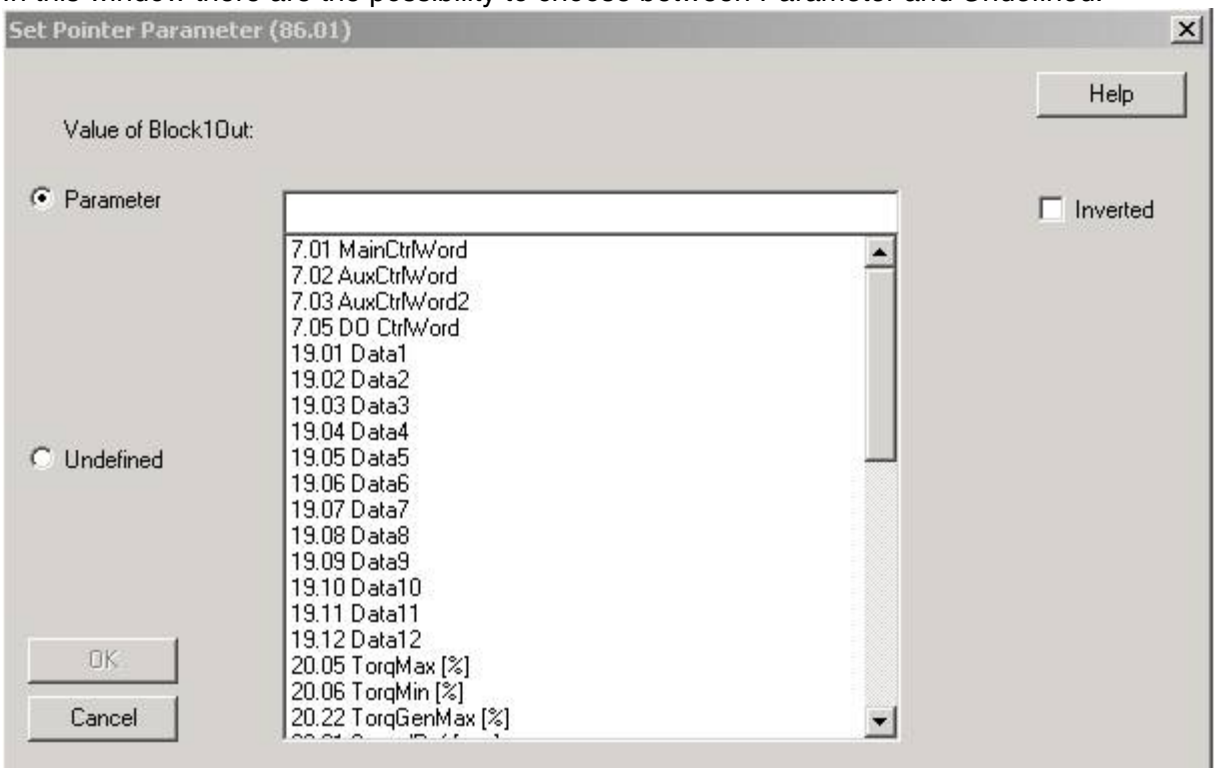
Edit output parameters

Pointer Parameter

All output write pointers will be shown in the right side of the desktop.
A click with Ctrl + left mouse button to the selected box opens a new window.



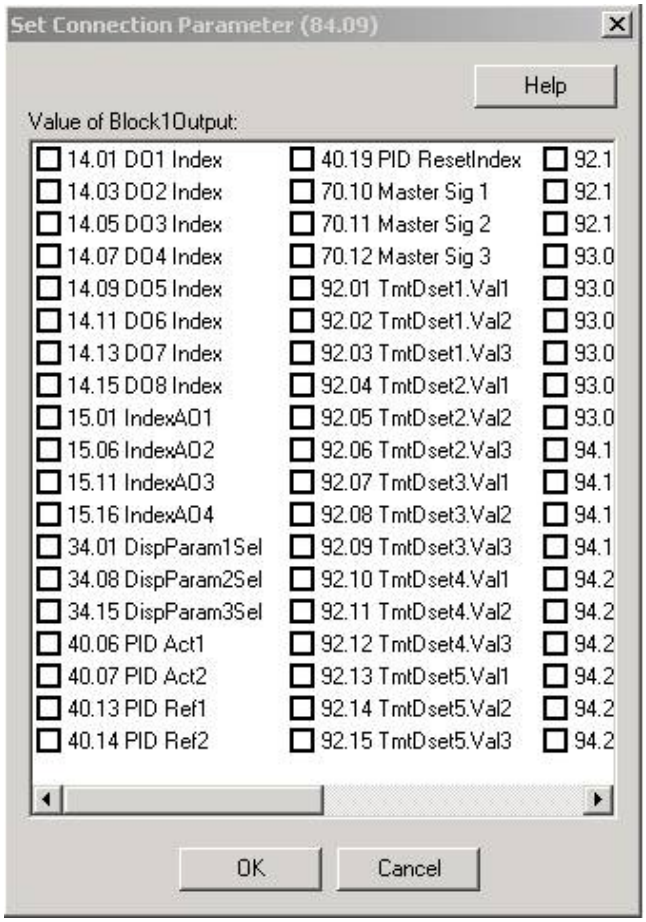
In this window there are the possibility to choose between *Parameter* and *Undefined*.



Parameter
Select a write address (parameter) from the list and click OK. There is the possibility to invert the output signal with a click to *Inverted*

Undefined
For disconnecting choose *Undefined*.

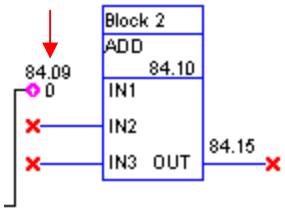
Output signals
It is possible to do a connection from the output of a block to the firmware of the drive. This is done via read functions. In this case select a read pointer (parameter) from the list and click into the check box. Multiple selections are possible!



Click OK to close the window.
You can see the choosen values at the desktop on the right side.

Input and output value viewing

You can see the actual value at each connection.
Click Shift + left mouse button to red connection point.



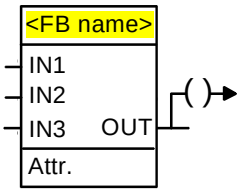
Then the blue line will be removed and the value will be shown.

Block Description

Function block details

General

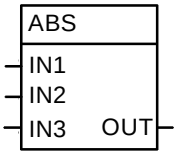
Each of the 16 function blocks has one up to max. three input parameters (group 84), which contains either an output address or a value of constant.
One further parameter is used for the attributes of these inputs. This attribute parameter is to be edited manually, if functions blocks are edited by using panel or by using parameter browser of DriveWindow (light).
By using Adaptive Programming PC tool this attribute parameter will be set automatically.
The output OUT, group 84, can be used for further inputs of function blocks. For writing the output value into standard parameters the output pointer, marked with -( )→, is to be set to the desired standard parameter. Output pointers can be found in group 86.



ABS

Type Arithmetic function

Illustration



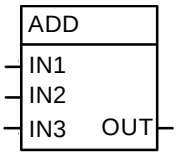
Operation The output is the absolute value of input IN1 multiplied by IN2 and divided by IN3.
OUT = |IN1| * IN2 / IN3

Connections
Input IN1, IN2 and IN3 : 16 bit integer values (15 bits + sign)
Output (OUT) : 16 bit integer (15 bits + sign)

ADD

Type Arithmetic function

Illustration



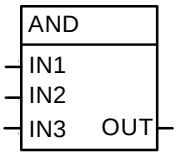
Operation The output is the sum of the inputs.
OUT = IN1 + IN2 + IN3

Connections
Input IN1, IN2 and IN3 : 16 bit integer values (15 bits + sign)
Output (OUT) : 16 bit integer (15 bits + sign)

AND

Type Logical function

Illustration



Operation The output is true if all connected inputs are true. Otherwise the output is false. Truth table:

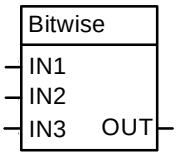
IN1	**IN2**	**IN3**	**OUT** (binary)	**OUT** (value on display)
0	0	0	False (All bits 0)	0
0	0	1	False (All bits 0)	0
0	1	0	False (All bits 0)	0
0	1	1	False (All bits 0)	0
1	0	0	False (All bits 0)	0
1	0	1	False (All bits 0)	0
1	1	0	False (All bits 0)	0
1	1	1	True (All bits 1)	-1

Connections
Input IN1, IN2 and IN3 : boolean values
Output (OUT) : 16 bit integer value (packed boolean)

Bitwise

Type Logical function

Illustration



Operation The block compares bits of three 16 bit word inputs and forms the output bits as follows:

OUT = (IN1 OR IN2) AND IN3.

Example, operation shown with only one bit:

IN1	IN2	IN3	OUT
0	0	0	0
0	1	0	0
1	0	0	0
1	1	0	0
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	1

Example, operation shown with whole word:

Input [word]		bits 15														0		Output [word]
20518	=> IN1	0	**1**	0	**1**	0	0	0	0	0	0	**1**	0	0	**1**	**1**	0	
4896	=> IN2	0	0	0	**1**	0	0	**1**	**1**	0	0	0	0	0	0	0	0	
17972	=> IN3	0	**1**	0	0	0	**1**	**1**	0	0	0	**1**	**1**	0	**1**	0	0	
		0	**1**	0	0	0	0	**1**	0	0	0	**1**	0	0	**1**	0	0	=> OUT 16932

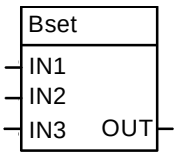
Connections

Input IN1, IN2 and IN3 : 16 bit integer values (packed boolean)
Output (OUT) : 16 bit integer values (packed boolean)

Bset

Type Logical function

Illustration



Operation Before the value of input IN1 will be set to the output (OUT), the bit number (IN2) of input word (IN1) will be set to the value of IN3.
Input IN1 is to be a packed word. The value of input IN2
IN3 should have the value 1 for true and 0 for false.

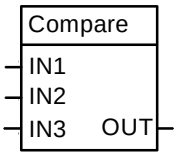
Connections

Input IN1 : packed 16-bit word
Input IN2 : 16 bit interger value, used 0 … 15 as bit number.
Input IN3 : boolean value
Output (OUT) : 16 bit packed word

Compare

Type Logical function

Illustration



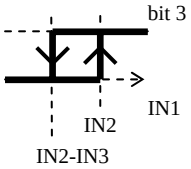
Operation

Output bits 0, 1 and 2:

- If IN1 > IN2, OUT = 001 Output bit 0 is true.
- If IN1 = IN2, OUT = 010 Output bit 1 is true.
- If IN1 < IN2, OUT = 100 Output bit 2 is true.

Output bit 3:

- If IN1 > IN2, OUT = 1ddd Output bit 3 is true and remains true until IN1 < (IN2 - IN3), after which bit 3 is false.



Output integer value, which is shown on display, is the sum of the bits :

bit 0	**bit 1**	**bit 2**	**bit 3**	**OUT** (value on display)
0	0	0	0	0
1	0	0	0	1
0	1	0	0	2
0	0	1	0	4
0	0	0	1	8
1	0	0	1	9
0	1	0	1	10
0	0	1	1	12

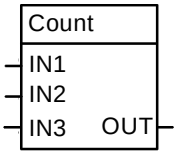
Connections

Input IN1, IN2 and IN3 : 16 bit integer values (15 bits + sign)
Output (OUT) : 16 bit integer (packed boolean)

Count

Type Arithmetic function

Illustration



Operation

The counter function counts rising edges of input IN1.
The counter is reset by the rising edges of input IN2 and limited to the value set with input IN3.

Input IN1 : Trigger (counter) input (0→1 edge)
Input IN2 : Reset input (0→1 edge).
Input IN3: : Max limit with value
- > 0: the output value increases up to max limit, which is the maximum.
- < 0: the output value increases up to the absolute value of max limit. With max limit the output will be set to 0 and starts countering with further trigger inputs.

Output (OUT) : The output shows the countered value.

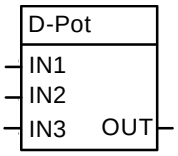
Connections

Input IN1, IN2 : Boolean values
Input IN3 : 16 bit integer value; 15 bit + sign
Output (OUT) : 15 bit integer value

D-Pot

Type Arithmetic function

Illustration



Operation

With input 1 the output will increase, with input 2 the output will decrease.
The absolute value of input 3 is the ramp time in ms related to 20000 of output. With positive sign of input 3 the output range is between 0 and 20000, with negative sign of input 3 the output range is between –20000 and +20000.
If both inputs 1 and 2 are active, input 2 (ramp down) will take action.

Input IN1 : Ramp up (bool)
Input IN2 : Ramp down (bool)
Input IN3 : ramp time, (ms rel. to 20000)
Output : 15+1 bit value

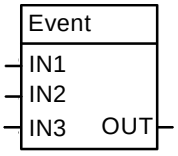
Connections

Input IN1 and IN2 : Boolean values
Input IN3 : 16 bit integer value; 15 bit + sign
Output (OUT) : 16 bit integer value; 15 bit + sign

Event

Type Viewing function

Illustration



Operation Input IN1 triggers the event. IN2 selects the number of fault, alarm, notice or trip texts. IN3 selects the type of the event (alarm, fault, notice or trip).

IN1	Activation input (boolean)		
	0->1	block activates the event	
	0	block deactivates the event	
IN2	Selection of displayed message. There exists 5 different messages, which are selected by using numbers depending on the type of event: The default message will be found in brackets.		
	Alarms	Faults and Trips	Notices
	301 (APAlarm1)	601 (APFault1)	801 ( )
	302 (APAlarm2)	602 (APFault2)	802 ( )
	303 (APAlarm3)	603 (APFault3)	803 ( )
	304 (APAlarm4)	604 (APFault4)	804 ( )
	305 (APAlarm5)	605 (APFault5)	805 ( )
IN3	Selection of type of event		
	0	Alarm ; shown as A30x	
	1	Fault ; shown as F60x. Faults have to be reset.	
	2	Notice, shown as N80x	
	3	Trip ; shown as fault F60x. A Trip will also open a connected DC breaker. Trips have to be reset.	

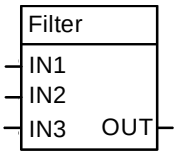
Connections

Input IN1 : 16 bit integer values (15 bits + sign)
Input IN2, IN3 : Selection of byte (compulsory)

Filter

Type Arithmetic function

Illustration



Operation The output is the filtered value of input IN1. Input IN2 is the filtering time.
$OUT = IN1 \cdot (1 - e^{-t/IN2})$
Note: The internal calculation uses 32 bits accuracy to avoid offset errors.

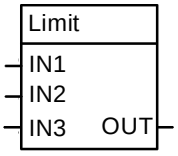
Connections

Input IN1 : 16 bit integer value (15 bits + sign)
Input IN2 : 16 bit integer value (15 bits + sign). One corresponds to 1 ms.
Output (OUT) : 16 bit integer (15 bits + sign)

Limit

Type Logical function

Illustration



Operation Value, connected to input IN1 will be limited with input IN2 as upper limit and with input IN3 as lower limit.
The output OUT makes the limeted input value available.
The output stays with 0, if the lower limit (input IN3) is greater or equal than the upper limit (input IN2).

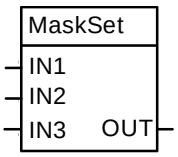
Connections

Input IN1, IN2 and IN3	: 16 bit integer value (15 bits + sign)
Output (OUT)	: 16 bit integer value (15 bits + sign)

MaskSet

Type Logical function

Illustration



Operation The block function sets or resets the bits defined in IN1 and IN2.

Input IN1: Word input
Input IN2: Set word input
Input IN3; Set/Reset IN2 in IN1.

Example, operation shown with only one bit:,

… with IN3 = Set

IN1	IN2	IN3	OUT
0	0	True	0
1	0	True	**1**
1	1	True	**1**
0	1	True	**1**

… with IN3 = Reset

IN1	IN2	IN3	OUT
0	0	False	0
1	0	False	**1**
1	1	False	0
0	1	False	0

Example, operation shown with whole word:

… with IN3 = true (=> Set)

Input [word]		bits 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0		Output [word]
26214	=> IN1	0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0		
-13108	=> IN2	1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0		
		1 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0	=> OUT	-4370

… with IN3 = false (=> Reset)

Input [word]		bits 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0		Output [word]
26214	=> IN1	0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0		
-13108	=> IN2	1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0		
		0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0	=> OUT	8738

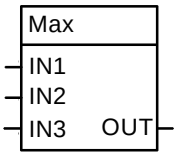
Connections

Input IN1 and IN2	: 16 bit integer value (packed boolean)
Input 3	: boolean
Output OUT	: 16 bit integer value (packed boolean)

Max

Type Arithmetic function

Illustration



Operation The output is the highest input value.
OUT = MAX (IN1, IN2, IN3)

Note: Open input will be taken as value zero.

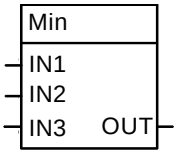
Connections

Input IN1, IN2 and IN3	:	16 bit integer values (15 bits + sign)
Output (OUT)	:	16 bit integer (15 bits + sign)

Min

Type Arithmetic function

Illustration



Operation The output is the lowest input value.
OUT = MIN (IN1, IN2, IN3)

Note: Open input will be taken as value zero.

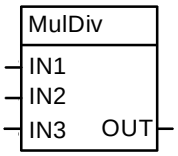
Connections

Input IN1, IN2 and IN3		16 bit integer values (15 bits + sign)
Output (OUT)	:	16 bit integer (15 bits + sign)

MulDiv

Type Arithmetic function

Illustration



Operation The output is the product of input IN1 and input IN2 divided by input IN3.
OUT = (IN1 · IN2) / IN3

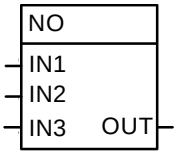
Connections

Input IN1, IN2 and IN3	:	16 bit integer values (15 bits + sign)
Output (OUT)	:	16 bit integer (15 bits + sign)

Not Used

Type -

Illustration



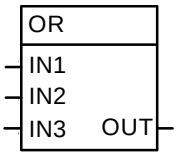
Operation Block is not enabled and not working (default setting).

Connections -

OR

Type Logical function

Illustration



Operation The output is true if any of the inputs is true. Truth table:

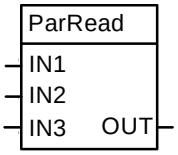
IN1	**IN2**	**IN3**	**OUT** (binary)	**OUT** (value on display)
0	0	0	False (All bits 0)	0
0	0	1	True (All bits 1)	-1
0	1	0	True (All bits 1)	-1
0	1	1	True (All bits 1)	-1
1	0	0	True (All bits 1)	-1
1	1	0	True (All bits 1)	-1
1	1	1	True (All bits 1)	-1

Connections
Input IN1, IN2 and IN3 : boolean values
Output (OUT) : 16 bit integer value (packed boolean)

ParRead

Type Logical function

Illustration



Operation Output (OUT) gives the value of a parameter, which is defined with input IN1 as parameter group and input IN2 as parameter index.

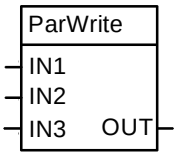
Example for reading parameter 22.01:
input IN1 = 22
input IN2 = 01

Connections
Input IN1 and IN2 : 16 bit integer value (15 bits + sign)
Output (OUT) : 16 bit integer value (15 bits + sign)

ParWrite

Type Logical function

Illustration



Operation Value of input IN1 is written into a parameter, which is defined with input IN2 as group x 100 + index.
Input IN3 can be set with a Boolean value: TRUE means save and FALSE means no save.

The output gives the error code, if parameter access is denided.

Example for parameter 22.01 = 150, not saving into FLASH.
input IN1 = the value of 150 (connection or constant)
input IN2 = 2201
input IN3 = false

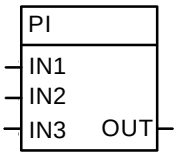
Connections

Input IN1 and IN2	: 16 bit integer value (15 bits + sign)
Input IN3	: Boolean value
Output OUT	: byte code

PI

Type Arithmetic controller

Illustration



Operation The output is input IN1 multiplied by IN2/100 plus integrated IN1 multiplied by IN3/100.

$$O = I1 * I2/100 + (I3/100) * \int I1$$

Note: The internal calculation uses 32 bits accuracy to avoid offset errors.

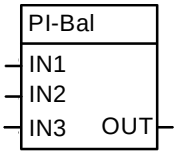
Connections

Input IN1	: 16 bit integer value (15 bit + sign)
Input IN2	: 16 bit integer value (15 bit + sign) Gain factor. 100 corresponds to 1.
Input IN3	: Integrator coefficient. 100 corresponds to 1. 10 000 corresponds to 100.
Output OUT	: 16 bit integer (15 bits + sign). The range is limited to 0 … 10000.

PI-Bal

Type Arithmetic function

Illustration



Operation The block initialises the PI block first. When input IN1 becomes true, the block writes the value of IN2 to the output of the PI block. When IN1 becomes false, the block releases the output of the PI controller block which continues normal operation from the set output.
Note: The block may be used only with the PI block. The block must follow the PI block.

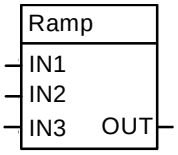
Connections

Input IN1	: boolean value
Input IN2	: 16 bit integer value (15 bits + sign)

Ramp

Type Arithmetic function

Illustration



Operation The block uses input IN1 as a reference value. With the ramp times (input IN2 and IN3) the output OUT increases or decreases as long as the reference value is reached.

Input IN1 :	Input value
Input IN2 :	Ramp up time, (ms, related to 20000)
Input IN3 :	Ramp down time, (ms, related to 20000)
Output :	integer output

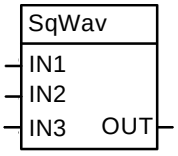
Connections

Input IN1	: 16 bit integer value; 15 bit + sign
Input IN2	: 16 bit integer value; 15 bit + sign
Input IN3	: 16 bit integer value; 15 bit + sign
Output OUT	: 16 bit integer value; 15 bit + sign

SqWav

Type Arithmetic function

Illustration



Operation The output OUT alternates between the value of input IN3 and zero (0), if the block is enabled with value of input IN1 = true.
The period is set with input IN2 with 1 = 1 ms.

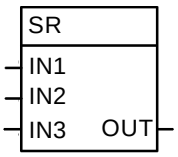
Connections

Input IN1	:	boolean value
Input IN2	:	16 bit integer value
Input IN3	:	16 bit integer value (15 bits + sign)
Output (OUT)	:	16 bit integer value (15 bits + sign)

SR

Type Logical function

Illustration



Operation Set/reset block. Input IN1 sets and IN2 and IN3 reset the output.
If IN1, IN2 and IN3 are false, the current value remains at the output.
If IN1 is true and IN2 and IN3 are false, the output is true.
If IN2 or IN3 is true, the output is false.

IN1	**IN2**	**IN3**	**OUT** (binary)	**OUT** (value on display)
0	0	0	Output	Output
0	0	1	False (All bits 0)	0
0	1	0	False (All bits 0)	0
0	1	1	False (All bits 0)	0
1	0	0	True (All bits 1)	-1
1	0	1	False (All bits 0)	0
1	1	0	False (All bits 0)	0
1	1	1	False (All bits 0)	0

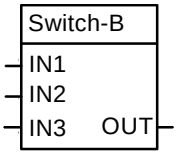
Connections

Input IN1, IN2 and IN3	:	boolean values
Output (OUT)	:	16 bit integer value (15 bits + sign)

Switch-B

Type Logical function

Illustration



Operation The output is equal to input IN2 if input IN1 is true and equal to input IN3 if input IN1 is false.

IN1			**OUT**	**OUT** (value on display)
0			= IN3	True = -1
1			= IN2	False = 0

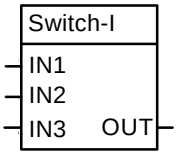
Connections

Input IN1, IN2 and IN3 : boolean values
Output (OUT) : 16 bit integer value (packed boolean)

Switch-I

Type Logical function

Illustration



Operation The output is equal to input IN2 if input IN1 is true and equal to input IN3 if input IN1 is false.

IN1			**OUT**
0			= IN3
1			= IN2

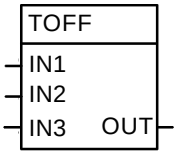
Connections

Input IN1 : boolean value
Input IN2 and IN3 : 16 bit integer values (15 bits + sign)
Output (OUT) : 16 bit integer value (15 bits + sign)

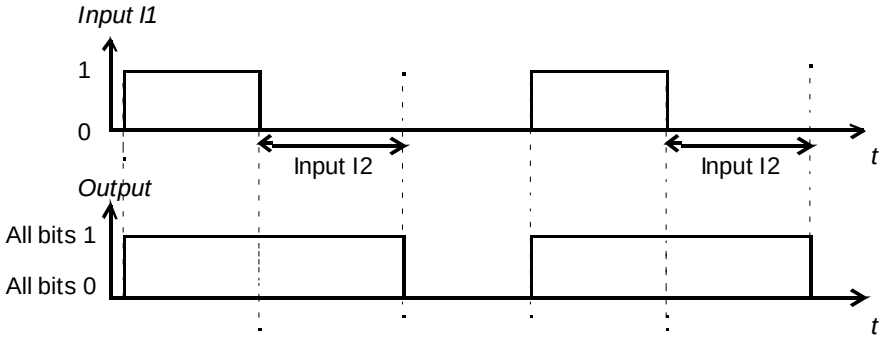
TOFF

Type Logical function

Illustration



Operation The output is true when input IN1 is true. The output is false when input IN1 has been false for a time equal or longer than input IN2.



Values on display: True = -1, false = 0.
With input 3 = False the delay time of input 2 is scaled in milliseconds (ms),
with input 3 = True the delay time of input 2 is scaled in seconds (s).

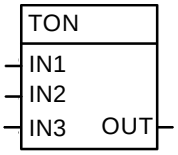
Connections

Input IN1 and IN3	: boolean value
Input IN2	: 16 bit integer value (15 bits + sign).
Output (OUT)	: 16 bit integer value (packed boolean)

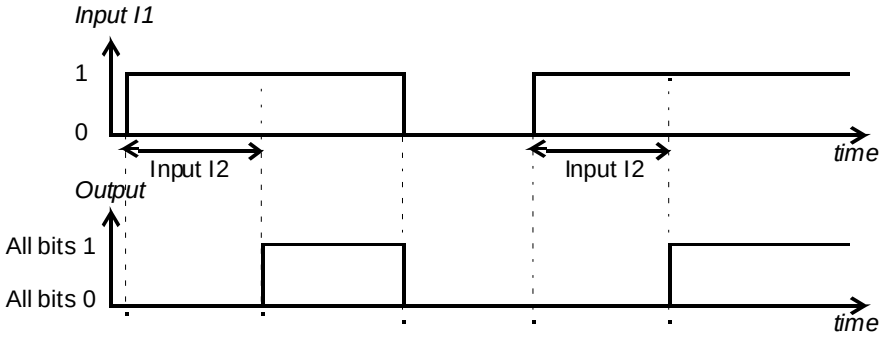
TON

Type Logical function

Illustration



Operation The output is true when input IN1 has been true for a time equal or longer than input IN2. The output is false when the input is false.



Values on display: True = -1, false = 0.
With input 3 = False the delay time of input 2 is scaled in milliseconds (ms),
with input 3 = True the delay time of input 2 is scaled in seconds (s).

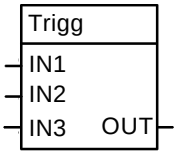
Connections

Input IN1 and IN3	: boolean value
Input IN2	: 16 bit integer value (15 bits + sign
Output (OUT)	: 16 bit integer value (packed boolean)

Trigg

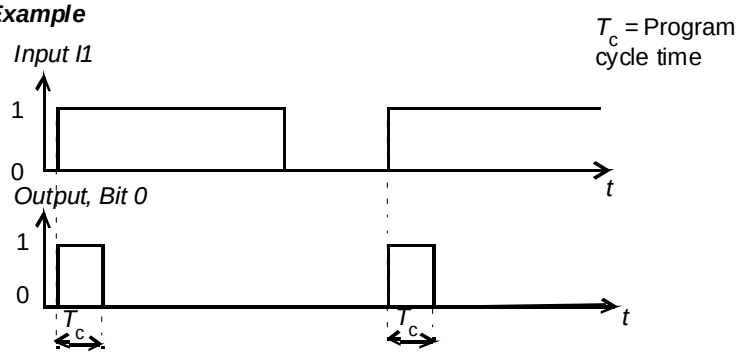
Type Logical function

Illustration



Operation The rising edge of input IN1 sets the output bit 0 for one program cycle.
The rising edge of input IN2 sets the output bit 1 for one program cycle.
The rising edge of input IN3 sets the output bit 2 for one program cycle.

Example

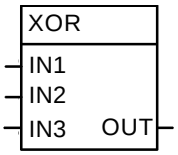


Connections
Input IN1, IN2 and IN3 : boolean values
Output (OUT) : 16 bit integer value (15 bits + sign)

XOR

Type Logical function

Illustration



Operation The output is true if one input is true, otherwise the output is false. Truth table:

IN1	**IN2**	**IN3**	**OUT** (binary)	**OUT** (value on display)
0	0	0	False (All bits 0)	0
0	0	1	True (All bits 1)	-1
0	1	0	True (All bits 1)	-1
0	1	1	False (All bits 0)	0
1	0	0	True (All bits 1)	-1
1	0	1	False (All bits 0)	0
1	1	0	False (All bits 0)	0
1	1	1	True (All bits 1)	-1

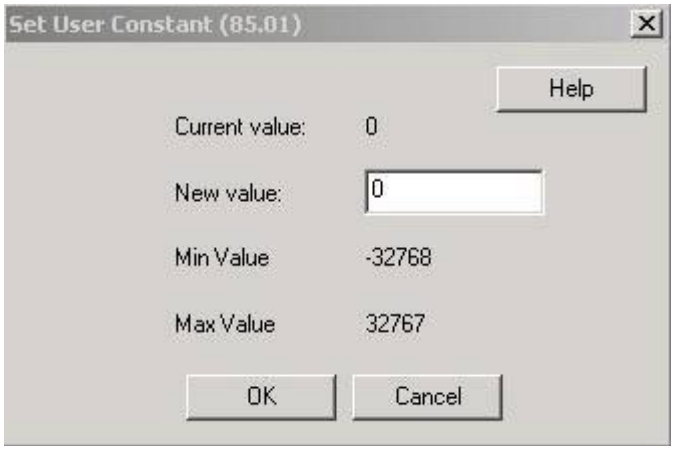
Connections
Input IN1, IN2 and IN3 : boolean values
Output (OUT) : 16 bit integer value (15 bits + sign)

Constants

There is a parameter group with constants for DWL AP programming. The values will be shown at the left side of the desktop.

Constants	
85.01	0
85.02	0
85.03	0
85.04	0
85.05	0
85.06	0
85.07	0
85.08	0
85.09	0
85.10	0

Click Ctrl + left mouse button to open the input window.



Put in a value and press OK.
The range for the input value is shown in the window.

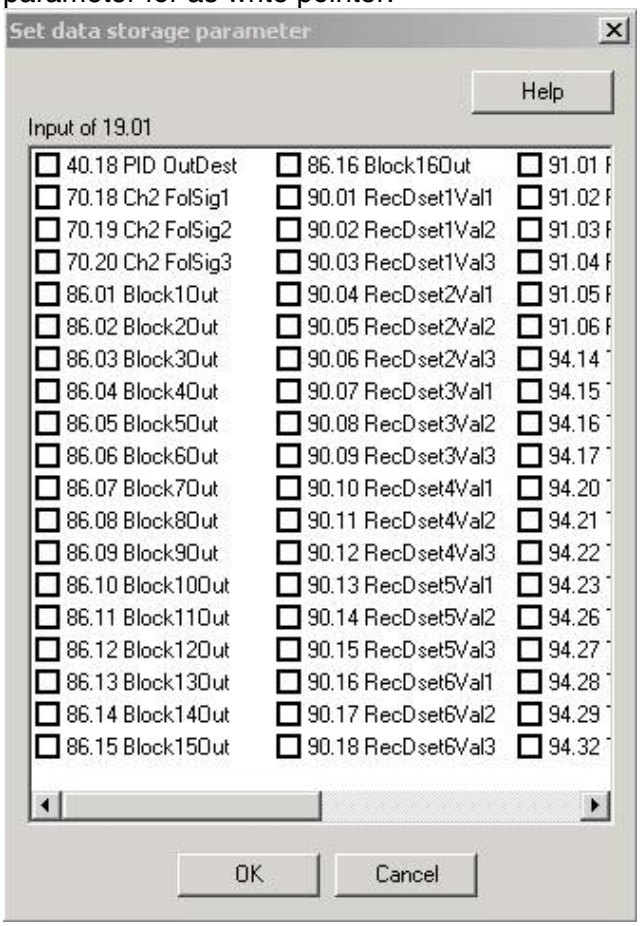
It is possible to connect an input of a function block with the parameter in group 85.

Data storage

The Data storage is a data container. It can be used for reading or writing signals and parameters. You can use parameter group 19 not only for DWL AP but also for CoDeSys, firmware and serial communication.
The actual value will be shown in the table.

Data Storage	
19.01	0
19.02	0
19.03	0
19.04	0
19.05	0
19.06	0
19.07	0
19.08	0
19.09	0
19.10	0
19.11	0
19.12	0

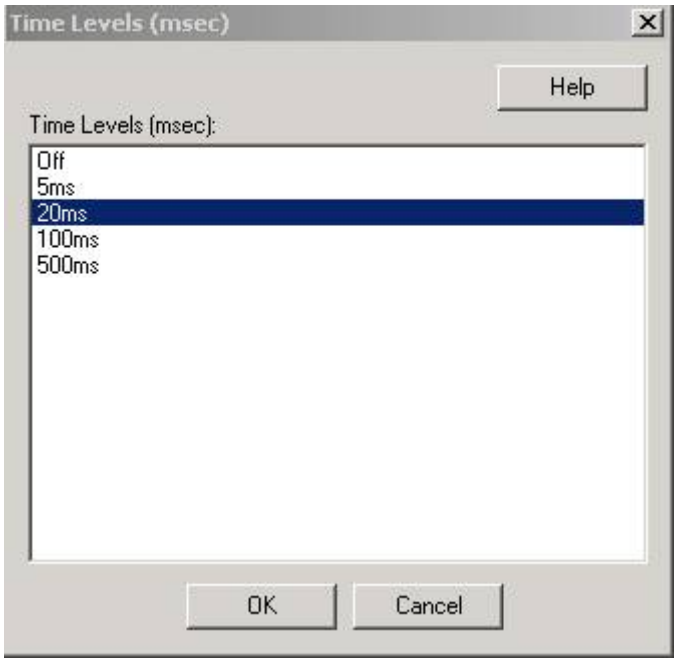
Click Ctrl + left mouse button to open the parameter window. In this window you can select the parameter for as write pointer.



A click on the parameter activate it and you can see a cross.

Time level

It is necessary to define a time level for the task (cycle time).
A click with Ctrl + left mouse button opens the Time level window.

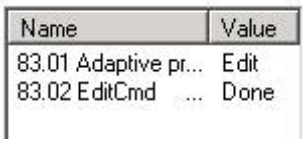


It is possible to select *5*, *20*, *100* and *500 ms*. If you selected *Off* there are a message, because the application is not working.

For Example:
A time level with 20 ms means that all connected in- and outputs were sampled every 20 milliseconds.

Adaptive Program Status

In the left side of the desktop you can see the actual drive status.
Parameter 83.01 shows the mode, e.g. *Start, Stop, Edit*, …
Parameter 83.02 shows the Edit Command, e.g. *Protect, Unprotect,* …



Parameter List (Group 83…86)

Group 83	Adaptive program control
83.01	**AdapProgCmd (Adaptive Program command)** Selects the operation mode for the adaptive Program: 0 = **Stop** stop, the Adaptive Program is not running and cannot be edited, default 1 = **Start** running, the Adaptive Program is running and cannot be edited 2 = **Edit** edit, the Adaptive Program is not running and can be edited 3 = **SingleCycle** The Adaptive Program runs only once. If a breakpoint is set with *BreakPoint (83.06)* the Adaptive Program will stop before the breakpoint. After the **SingleCycle** *AdapProgCmd (83.01)* is automatically set back to **Stop**. 4 = **SingleStep** Runs only one function block. *LocationCounter (84.03)* shows the function block number, which will be executed during the next **SingleStep**. After a **SingleStep** *AdapProgCmd (83.01)* is automatically set back to **Stop**. *LocationCounter (84.03)* shows the next function block to be executed. To reset *LocationCounter (84.03)* to the first function block set *AdapProgCmd (83.01)* to **Stop** again (even if it is already set to **Stop**). **Note1:** *AdapProgCmd (83.01)* = **Start**, **SingleCycle** or **SingleStep** is only valid, if *AdapPrgStat (84.01)* **Running**. **Int. Scaling: 1 == 1** **Type: C** **Volatile: N**
83.02	**EditCmd (edit command)** Edit application program. *EditCmd (83.02)* is automatically set back to **Done** after the chosen action is finished: 0 = **Done** no action or edit application program completed, default 1 = **Push** Shifts the function block in the spot defined by *EditBlock (83.03)* and all subsequent function blocks one spot forward. A new function block can be placed in the now empty spot by programming its parameter set as usual. Example: A new function block needs to be placed in between the function block number four (84.22) to (84.27) and five (84.28) to (84.33). In order to do this: 1. set *AdapProgCmd (83.01)* = **Edit** 2. set *EditBlock (83.03)* = 5 (selects function block 5 as the desired spot for the new function block) 3. set *EditCmd (83.02)* = **Push** (shifts function block 5 and all subsequent function blocks one spot forward) 4. Program empty spot 5 by means of (84.28) to (84.33) 2 = **Delete** Deletes the function block in the spot defined by *EditBlock (83.03)* and shifts all subsequent function blocks one spot backward. To delete all function blocks set *EditBlock (83.03)* = 17. 3 = **Protect** Turns all parameters of the Adaptive Program into protected mode (parameters cannot be written to). Before using the **Protect** command set the pass code by means of *PassCode (83.05).* **Attention:** Do not forget the pass code! 4 = **Unprotect** Reset of protected mode. Before the **Unprotect** command can be used, *PassCode (83.05)* has to be set. **Attention:** The proper pass code has to be used! **Int. Scaling: 1 == 1** **Type: C** **Volatile: Y**
83.03	**EditBlock (edit block)** Defines the function block witch is selected by *EditCmd (83.02)* = **Push** or **Delete**. After a **Push** or **Delete** *EditBlock (83.03)* is automatically set back to 1. **Note1:** To delete all function blocks set *EditBlock (83.03)* = 17. **Int. Scaling: 1 == 1** **Type: I** **Volatile: Y**

83.04 **TimeLevSel (time level select)**
Selects the cycle time for the Adaptive Program. This setting is valid for all function blocks.
0 = **Off** no task selected
1 = **5ms** Adaptive Program runs with 5 ms
2 = **20ms** Adaptive Program runs with 20 ms
3 = **100ms** Adaptive Program runs with 100 ms
4 = **500ms** Adaptive Program runs with 500 ms
Int. Scaling: 1 == 1 Type: C Volatile: N

83.05 **PassCode (pass code)**
The pass code is a number between 1 and 65535 to write protect Adaptive Programs by means of *EditCmd (83.02)*. After using **Protect** or **Unprotect** PassCode (83.05) is automatically set back to zero.
Attention:
Do not forget the pass code!
Int. Scaling: 1 == 1 Type: I Volatile: Y

83.06 **BreakPoint (break point)**
Breakpoint for *AdapProgCmd (83.01)* = **SingleCycle**.
The break point is not used, if *BreakPoint (83.06)* is set to zero.
Int. Scaling: 1 == 1 Type: I Volatile: Y

Group 84

Adaptive program

84.01 **AdapPrgStat (Adaptive Program status word)**
Adaptive program status word:

Bit	Name	Value	Comment
B0	**Bit 0**	1	Adaptive Program is running
		0	Adaptive Program is stopped
B1	**Bit 1**	1	Adaptive Program can be edited
		0	Adaptive Program cannot be edited
B2	**Bit 2**	1	Adaptive Program is being checked
		0	no action
B3	**Bit 3**	1	Adaptive Program is faulty
		0	Adaptive Program is OK

Faults in the Adaptive Program can be:
used function block with not at least input 1 connection
used pointer is not valid
invalid bit number for function block **Bset**
location of function block **PI-Bal** after **PI** function block
Int. Scaling: 1 == 1 Type: I Volatile: Y

84.02 **FaultedPar (faulted parameters)**
The Adaptive Program will be checked before running. If there is a fault, *AdapPrgStat (84.01)* is set to “faulty” and FaultedPar (84.02) shows the faulty input.
Note1:
In case of a problem check the value and the attribute of the faulty input.
Int. Scaling: 1 == 1 Type: I Volatile: Y

84.03 **LocationCounter (location counter)**
Location counter for *AdapProgCmd (83.01)* = **SingleStep** shows the function block number, which will be executed next.
Int. Scaling: 1 == 1 Type: I Volatile: Y

84.04 **Block1Type (function block 1 type)**

Selects the type for function block 1 [Block Parameter Set 1 (BPS1)]. Detailed description of the type can be found in chapter '*Function blocks*':

0 = **NotUsed**	function block is not used
1 = **ABS**	absolute value
2 = **ADD**	sum
3 = **AND**	AND
4 = **Bitwise**	bit compare
5 = **Bset**	bit set
6 = **Compare**	compare
7 = **Count**	counter
8 = **D-Pot**	ramp
9 = **Event**	event
10 = **Filter**	filter
11 = **Limit**	limit
12 = **MaskSet**	mask set
13 = **Max**	maximum
14 = **Min**	minimum
15 = **MulDiv**	multiplication and division
16 = **OR**	OR
17 = **ParRead**	parameter read
18 = **ParWrite**	parameter write
19 = **PI**	PI-controller
20 = **PI-Bal**	initialization for PI-controller
21 = **Ramp**	ramp
22 = **SqWav**	square wave
23 = **SR**	SR flip-flop
24 = **Switch-B**	switch Boolean
25 = **Switch-I**	switch integer
26 = **TOFF**	timer off
27 = **TON**	timer on
28 = **Trigg**	trigger
29 = **XOR**	exclusive OR

Int. Scaling: 1 == 1 **Type: C** **Volatile: N**

84.05 **Block1In1 (function block 1 input 1)**

Selects the source for input 1 of function block 1 (BPS1). There are 2 types of inputs, signals/parameters and constants:

Signals/parameters are all signals and parameters available in the drive. The format is **-xxyy**, with: **-** = negate signal/parameter, **xx** = group and **yy** = index.
Example:
To connect negated *SpeedRef (23.01)* set *Block1In1 (84.05)* = -2301 and *Block1Attrib (84.08)* = 0h.
To get only a certain bit e.g. **RdyRef** bit 3 of *MainStatWord (8.01)* set *Block1In1 (84.05)* = 801 and *Block1Attrib (84.08)* = 3h.
Constants are feed directly into the function block input and have to be declared by means of *Block1Attrib (84.08).*
Example:
To connect the constant value of 12345 set *Block1In1 (84.05)* = 12345 and *Block1Attrib (84.08)* = 1000h.

Int. Scaling: 1 == 1 **Type: SI** **Volatile: N**

84.06 **Block1In2 (function block 1 input 2)**

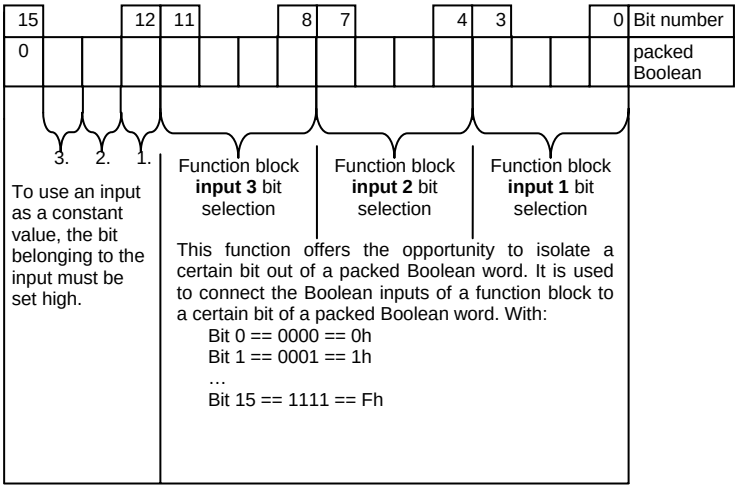
Selects the source for input 2 of function block 1 (BPS1). Description see *Block1In1 (84.05)*, except:
Example:
To get only a certain bit e.g. **RdyRef** bit 3 of *MainStatWord (8.01)* set *Block1In2 (84.06)* = 801 and *Block1Attrib (84.08)* = 30h.

Int. Scaling: 1 == 1 **Type: SI** **Volatile: N**

84.07 **Block1In3 (function block 1 input 3)**
Selects the source for input 2 of function block 1 (BPS1). Description see *Block1In1 (84.05)*, except:
Example:
To get only a certain bit e.g. **RdyRef** bit 3 of *MainStatWord (8.01)* set *Block1In3 (84.07)* = 801 and *Block1Attrib (84.08)* = 300h.
Int. Scaling: 1 == 1 **Type:** **SI** **Volatile: N**

84.08 **Block1Attrib (function block 1 attribute)**
Defines the attributes of function block 1 for all three inputs [*Block1In1 (84.05)*, *Block1In2 (84.06)* and *Block1In3 (84.07)*] (BPS1).
Block1Attrib (84.08) is divided into 4 parts:

Bit number 0 - 3 for input 1 to get a certain bit out of a packed Boolean word.
Bit number 4 - 7 for input 2 to get a certain bit out of a packed Boolean word.
Bit number 8 - 11 for input 3 to get a certain bit out of a packed Boolean word.
Bit number 12 - 14 for input 1 - 3 to feed a constant directly into the input



Int. Scaling: 1 == 1 **Type:** **h** **Volatile: N**

84.09 **Block1Output (function block 1 output)**
Function block 1 output, can be used as an input for further function blocks.
Int. Scaling: 1 == 1 **Type:** **SI** **Volatile: Y**

84.10 to 84.99 The description of the parameters for function blocks 2 to 16 is basically the same as for function block 1. For Your convenience the following table shows the parameter numbers of all function blocks1:

Function block	BlockxType	BlockxIn1 input 1	BlockxIn2 input 2	BlockxIn3 input 1	BlockxAttrib	BlockxOutput signal	BlockxOut pointer
1	84.04	84.05	84.06	84.07	84.08	84.09	86.01
2	84.10	84.11	84.12	84.13	84.14	84.15	86.02
3	84.16	84.17	84.18	84.19	84.20	84.21	86.03
4	84.22	84.23	84.24	84.25	84.26	84.27	86.04
5	84.28	84.29	84.30	84.31	84.32	84.33	86.05
6	84.34	84.35	84.36	84.37	84.38	84.39	86.06
7	84.40	84.41	84.42	84.43	84.44	84.45	86.07
8	84.46	84.47	84.48	84.49	84.50	84.51	86.08
9	84.52	84.53	84.54	84.55	84.56	84.57	86.09
10	84.58	84.59	84.60	84.61	84.62	84.63	86.10
11	84.64	84.65	84.66	84.67	84.68	84.69	86.11
12	84.70	84.71	84.72	84.73	84.74	84.75	86.12
13	84.76	84.77	84.78	84.79	84.80	84.81	86.13
14	84.82	84.83	84.84	84.85	84.86	84.87	86.14
15	84.88	84.89	84.90	84.91	84.92	84.93	86.15
16	84.94	84.95	84.96	84.97	84.98	84.99	86.16

Group 85	User constants
85.01	**Constant1 (constant 1)** Sets an integer constant for the Adaptive Program. **Int. Scaling: 1 == 1 Type: SI Volatile: N**
85.02	**Constant2 (constant 2)** Sets an integer constant for the Adaptive Program. **Int. Scaling: 1 == 1 Type: SI Volatile: N**
85.03	**Constant3 (constant 3)** Sets an integer constant for the Adaptive Program. **Int. Scaling: 1 == 1 Type: SI Volatile: N**
85.04	**Constant4 (constant 4)** Sets an integer constant for the Adaptive Program. **Int. Scaling: 1 == 1 Type: SI Volatile: N**
85.05	**Constant5 (constant 5)** Sets an integer constant for the Adaptive Program. **Int. Scaling: 1 == 1 Type: SI Volatile: N**
85.06	**Constant6 (constant 6)** Sets an integer constant for the Adaptive Program. **Int. Scaling: 1 == 1 Type: SI Volatile: N**
85.07	**Constant7 (constant 7)** Sets an integer constant for the Adaptive Program. **Int. Scaling: 1 == 1 Type: SI Volatile: N**
85.08	**Constant8 (constant 8)** Sets an integer constant for the Adaptive Program. **Int. Scaling: 1 == 1 Type: SI Volatile: N**
85.09	**Constant9 (constant 9)** Sets an integer constant for the Adaptive Program. **Int. Scaling: 1 == 1 Type: SI Volatile: N**
85.10	**Constant10 (constant 10)** Sets an integer constant for the Adaptive Program. **Int. Scaling: 1 == 1 Type: SI Volatile: N**
85.11	**String1 (string 1)** Sets a string for the Adaptive Program. With DriveWindow it is possible to fill in a string (e.g. name of an event) with a maximum of 12 characters. This string is shown in the control panel and in DriveWindow. **Int. Scaling: 1 == 1 Type: SI/C Volatile: N**
85.12	**String2 (string 2)** Sets a string for the Adaptive Program. With DriveWindow it is possible to fill in a string (e.g. name of an event) with a maximum of 12 characters. This string is shown in the control panel and in DriveWindow. **Int. Scaling: 1 == 1 Type: SI/C Volatile: N**

85.13	**String3 (string 3)** Sets a string for the Adaptive Program. With DriveWindow it is possible to fill in a string (e.g. name of an event) with a maximum of 12 characters. This string is shown in the control panel and in DriveWindow. **Int. Scaling: 1 == 1 Type: SI/C Volatile: N**
85.14	**String4 (string 4)** Sets a string for the Adaptive Program. With DriveWindow it is possible to fill in a string (e.g. name of an event) with a maximum of 12 characters. This string is shown in the control panel and in DriveWindow. **Int. Scaling: 1 == 1 Type: SI/C Volatile: N**
85.15	**String5 (string 5)** Sets a string for the Adaptive Program. With DriveWindow it is possible to fill in a string (e.g. name of an event) with a maximum of 12 characters. This string is shown in the control panel and in DriveWindow. **Int. Scaling: 1 == 1 Type: SI/C Volatile: N**
Group 86	**Adaptive program outputs**
86.01	**Block1Out (block 1 output)** The value of function block 1 output [*Block1Output (84.09)*] is written to a sink (signal/parameter) by means of this index pointer [e.g. 2301 equals *SpeedRef (23.01)*]. The format is **-xxyy**, with: **-** = negate signal/parameter, **xx** = group and **yy** = index. **Int. Scaling: 1 == 1 Type: I Volatile: N**
86.02	**Block2Out (block 2 output)** The value of function block 2 output [*Block2Output (84.15)*] is written to a sink (signal/parameter) by means of this index pointer [e.g. 2301 equals *SpeedRef (23.01)*]. The format is **-xxyy**, with: **-** = negate signal/parameter, **xx** = group and **yy** = index. **Int. Scaling: 1 == 1 Type: I Volatile: N**
86.03	**Block3Out (block 3 output)** The value of function block 3 output [*Block3Output (84.21)*] is written to a sink (signal/parameter) by means of this index pointer [e.g. 2301 equals *SpeedRef (23.01)*]. The format is **-xxyy**, with: **-** = negate signal/parameter, **xx** = group and **yy** = index. **Int. Scaling: 1 == 1 Type: I Volatile: N**
86.04	**Block4Out (block 4 output)** The value of function block 4 output [*Block1Output (84.27)*] is written to a sink (signal/parameter) by means of this index pointer [e.g. 2301 equals *SpeedRef (23.01)*]. The format is **-xxyy**, with: **-** = negate signal/parameter, **xx** = group and **yy** = index. **Int. Scaling: 1 == 1 Type: I Volatile: N**
86.05	**Block5Out (block 5 output)** The value of function block 5 output [*Block1Output (84.33)*] is written to a sink (signal/parameter) by means of this index pointer [e.g. 2301 equals *SpeedRef (23.01)*]. The format is **-xxyy**, with: **-** = negate signal/parameter, **xx** = group and **yy** = index. **Int. Scaling: 1 == 1 Type: I Volatile: N**
86.06	**Block6Out (block 6 output)** The value of function block 6 output [*Block1Output (84.39)*] is written to a sink (signal/parameter) by means of this index pointer [e.g. 2301 equals *SpeedRef (23.01)*]. The format is **-xxyy**, with: **-** = negate signal/parameter, **xx** = group and **yy** = index. **Int. Scaling: 1 == 1 Type: I Volatile: N**
86.07	**Block7Out (block 7 output)** The value of function block 7 output [*Block1Output (84.45)*] is written to a sink (signal/parameter) by means of this index pointer [e.g. 2301 equals *SpeedRef (23.01)*]. The format is **-xxyy**, with: **-** = negate signal/parameter, **xx** = group and **yy** = index. **Int. Scaling: 1 == 1 Type: I Volatile: N**

86.08	**Block8Out (block 8 output)** The value of function block 8 output [*Block1Output (84.51)*] is written to a sink (signal/parameter) by means of this index pointer [e.g. 2301 equals *SpeedRef (23.01)*]. The format is **-xxyy**, with: **-** = negate signal/parameter, **xx** = group and **yy** = index. **Int. Scaling: 1 == 1 Type: I Volatile: N**
86.09	**Block9Out (block 9 output)** The value of function block 9 output [*Block1Output (84.57)*] is written to a sink (signal/parameter) by means of this index pointer [e.g. 2301 equals *SpeedRef (23.01)*]. The format is **-xxyy**, with: **-** = negate signal/parameter, **xx** = group and **yy** = index. **Int. Scaling: 1 == 1 Type: I Volatile: N**
86.10	**Block10Out (block 10 output)** The value of function block 10 output [*Block1Output (84.63)*] is written to a sink (signal/parameter) by means of this index pointer [e.g. 2301 equals *SpeedRef (23.01)*]. The format is **-xxyy**, with: **-** = negate signal/parameter, **xx** = group and **yy** = index. **Int. Scaling: 1 == 1 Type: I Volatile: N**
86.11	**Block11Out (block 11 output)** The value of function block 11 output [*Block1Output (84.69)*] is written to a sink (signal/parameter) by means of this index pointer [e.g. 2301 equals *SpeedRef (23.01)*]. The format is **-xxyy**, with: **-** = negate signal/parameter, **xx** = group and **yy** = index. **Int. Scaling: 1 == 1 Type: I Volatile: N**
86.12	**Block12Out (block 12 output)** The value of function block 12 output [*Block1Output (84.75)*] is written to a sink (signal/parameter) by means of this index pointer [e.g. 2301 equals *SpeedRef (23.01)*]. The format is **-xxyy**, with: **-** = negate signal/parameter, **xx** = group and **yy** = index. **Int. Scaling: 1 == 1 Type: I Volatile: N**
86.13	**Block13Out (block 13 output)** The value of function block 13 output [*Block1Output (84.81)*] is written to a sink (signal/parameter) by means of this index pointer [e.g. 2301 equals *SpeedRef (23.01)*]. The format is **-xxyy**, with: **-** = negate signal/parameter, **xx** = group and **yy** = index. **Int. Scaling: 1 == 1 Type: I Volatile: N**
86.14	**Block14Out (block 14 output)** The value of function block 14 output [*Block1Output (84.87)*] is written to a sink (signal/parameter) by means of this index pointer [e.g. 2301 equals *SpeedRef (23.01)*]. The format is **-xxyy**, with: **-** = negate signal/parameter, **xx** = group and **yy** = index. **Int. Scaling: 1 == 1 Type: I Volatile: N**
86.15	**Block15Out (block 15 output)** The value of function block 15 output [*Block1Output (84.93)*] is written to a sink (signal/parameter) by means of this index pointer [e.g. 2301 equals *SpeedRef (23.01)*]. The format is **-xxyy**, with: **-** = negate signal/parameter, **xx** = group and **yy** = index. **Int. Scaling: 1 == 1 Type: I Volatile: N**
86.16	**Block16Out (block 16 output)** The value of function block 16 output [*Block16Output (84.99)*] is written to a sink (signal/parameter) by means of this index pointer [e.g. 2301 equals *SpeedRef (23.01)*]. The format is **-xxyy**, with: **-** = negate signal/parameter, **xx** = group and **yy** = index. **Int. Scaling: 1 == 1 Type: I Volatile: N**

ABB Automation Products
Wallstadter Str. 59
68526 Lampertheim • Germany
Phone: +49 (0) 62 03-71-0
Fax: +49 (0) 62 03-71-7609
www.abb.com/motors&drives

Ident. No.: 3ADW 000 209 R0101 Rev A
02_2006

209R0101A6080000