Engineer^{IT}
Control Builder F

Engineering Manual
IEC 61131-3 Programming

Notice

Information provided in this manual is subject to change without prior notice and represents no obligation on the part of ABB Automation Products.

The industrial standards and regulations (e.g. DIN, VDE, VDI, etc.) applicable in the Federal Republic of Germany are used. Outside the Federal Republic of Germany, the relevant national specifications, standards and regulations must be observed.

The designations used and the products shown/mentioned in this manual have not been specifically marked regarding existing industrial property rights.

Should you find any mistakes in this manual, please make a copy of the appropriate page(s) and send it/them to us with your comments. Any suggestions which may help to improve comprehension or clarity will also be gratefully accepted.

Please send your suggestions to:

Product Management Dept., DEAPR/LMS-Hannover, Fax: +49 (0)511 6782 701

Engineering Manual
IEC 61131-3 Programming

Engineering Manuals

System Configuration

A General Information
B Installation DigiTool
C Project Manager
D Project Tree
E Hardware Structure
F Commissioning
G Documentation
Glossary
New Features
Index

IEC 61131-3 Programming

A General Information
B Variables
C Tags
D Function Block Diagram (FBD)
E Instruction List (IL)
F Ladder Diagram (LD)
G Sequential Function Chart (SFC)
H User Function Blocks
Index

Operator Station

A General Information
B Messages and Hints
C Standard Displays
D Graphic Display
E Logs
Index

Engineering Manuals

Process Station – Rack based System

1 General
2 Loading the operating system and EPROM'S
3 Configuration of resource D-PS and D-PS/RED in project tree
4 Processing and failure action
5 Configuration of rack-based process station in the hardware structure
6 Commissioning the process station
7 Redundancy

Process Station –FieldController

1 General
2 Loading the operating system and EPROM'S
3 Configuration of resource D-PS in project tree
4 Processing and failure action
5 Configuration of process station FC in the hardware structure
6 Commissioning of FC

Process Station – ABB FieldController 800

1 General
2 Loading the operating system and EPROM'S
3 Configuration of resource D-PS and D-PS/RED in project tree
4 Processing and failure action
5 Configuration of process station AC 800F in the hardware structure
6 Commissioning of AC 800F
7 Redundancy

Engineering Reference Manuals

Functions and Function Blocks

A	Getting Started
B	General Description and Overview
C	Analog Function Blocks
D	Binary Function Blocks
E	Controller Function Blocks
F	Acquisition Blocks
G	Monitoring Function Blocks
H	Open-loop Control Function Blocks
J	Standard Function Blocks
K	Arithmetic Blocks
L	Converter Blocks
M	Constants
N	System Functions
V	Abbreviations
W	Glossary
X	Index

Communications and Fieldbusses

A	Getting Started
B	General Description and Overview
C	Profibus
E	Modbus Master Function Blocks
F	Modbus Slave Function Blocks
G	Send / Receive Blocks
H	Rack Modules
V	Abbreviations
W	Glossary
X	Index

Operators Manual

Operator Station

A General Information

B Variables

C Tags

D Function Block Diagram (FBD)

E Instruction List (IL)

F Ladder Diagram

G Sequential Function Chart (SFC)

H User Function Blocks

X Index

A General Information

Contents

A 1 Visual Orientation Hints

To grant direct access to information, we have used different types of scripts and symbols.

Script	Meaning
Italics	Representation for (selectable) menu items or parameters.
SMALL CAPITALS	Inputs to be made via the keyboard, also via virtual keys.
boldface	Highlights important information, also as an orientation hint.

Symbols	Meaning
	Selection with mouse. The various instruction steps are separated by arrows. Example: → *Edit* → *Dimensions* → ... . In this example, the menu item *Edit* is to be selected followed by the menu item *Dimensions*.
	Operating alternative with the mouse
	Information on operation with the keyboard, inasmuch as it differs from the Windows Standard. Example: Select module → ALT → E → D → ... . Having selected the module via the ARROW KEYS, the keys ALT, E and D must be pressed successively. If two keys are to be pressed simultaneously: ... → SHIFT + INSERT → ... .
	Alternative keyboard operation
	Hints
	Special hints, must be observed!
Preconditions	Preconditions which must be fulfilled to implement commands or for satisfactory results.

A 2 General Hints for Operating DigiTool

Operation of DigiTool is based on the Windows Standard.
Therefore knowledge of general operation under Windows is strongly recommended, see Windows Manual.

The "typical Windows operation" will therefore not be dealt with in detail when describing the various editors.

A 2.1 Operation with mouse or keyboard

	Mouse 	Keyboard 
Select menu items	Cursor on menu item + left mouse button.	ALT + underlined letters
Select within pull-down menu	Cursor on menu item + left mouse button.	Enter only underline letters
Select individual elements	Cursor on program element + left mouse button	Cursor on program element + SPACE
Select multiple elements	Cursor on start position → press left mouse button and keep pressed down → move to desired position and release mouse button	Cursor on start position → Press SPACE and hold → move to desired position and release SPACE

A 2.2 Recurring keys

di0212uk.bmp

OK	The active parameter window is quit and the parameter status **saved**.
CANCEL	The active parameter window is quit **without saving** the parameter status. A warning appears if parameter definition data are lost.
SAVE	The current parameter status is **saved** and the window remains active.
RESET	The parameters of the active parameter window are reset completely to the previously **set values**. Any parameters previously saved and differing from the default settings can be fetched again by canceling and recalling the parameter window.
CHECK	The function block is checked for plausibility with the current parameters, even if they have not been saved.
HELP	Call up the Windows On-line help (with F1 also). To get information about functions on monitor without using the documentation. Thematically structured information is displayed in the help-window.

<< >>	Change to the previous or next parameter mask. This is displayed with shading if no further parameter window exist.
☑ ☐	Check boxes A setting or parameter is turned on or off.
○ ◉	Option fields Option fields are presented when **one** of a group of mutually parameters are to be chosen.
F5	The function key F5 calls the list of cross references for the selected variable or tag. This function is also available in parameter mask fields with referenced variables or tags.
F6 bzw. SHIFT+F6	These functions are available after the list of all existing cross references was called (key F5). F6 calls the program which contains the next occurrence in the cross reference list, SHIFT+F6 calls the program of the previous list entry.

B Variables

Name	Comment	Type	D-PS	X	Object	Location	P
AB01		BOOL	PS00	Y	DDI01	PS_1_1_0_1	N
AI1B1N		BOOL	PS00	N			Y
AI1B2N		BOOL	PS00	N			Y
AI1B3N		BOOL	PS00	N			Y
AI1B4N		BOOL	PS00	N			Y
AI1B5N		BOOL	PS00	N			Y
AI1_DEVICE_ERROR		BOOL	PS00	N			Y
AI1_HIGH_LIMIT		BOOL	PS00	N			Y
AI1_LOW_LIMIT		BOOL	PS00	N			Y
AI1_NO_COMM		BOOL	PS00	N			Y
AI1_NO_COMM_AKT		BOOL	PS00	N			Y
AI1_OUT_OF_RANGE		BOOL	PS00	N			Y
AI1_SUMMEN_ST		BOOL	PS00	N			Y
AI2B1N		BOOL	PS00	N			Y
AI2B2N		BOOL	PS00	N			Y

Contents

B 1 General Description - Variables

Variables are used for storing and processing information. Various different data types are available in the system, e.g. BYTE, WORD, INTEGER, REAL, DATA&TIME. To enable several variables to be processed jointly even if they have different data types, it is possible to define new, **structured data types**.

Along with the standard data types, user-defined structured data types are also available when declaring a variable.

System variables are created every time a new resource, process station, FieldController or gateway is added. The status details for the resource are stored in these variables.

The system enters all the variables for a project in the **variable list**.

Default values can be assigned to each variable and to the separate elements of a structured variable. These values are assumed after a cold start, or when a station is initialized.

Variables from Freelance 2000 can be made available to other systems via gateway stations. For this purpose, read/write accesses are configured in the station view of the variables list.

☞ **From version 5 onwards** it is possible to use variables with leading numbers, e.g. 2LAB10CF001. A variable name must contain at least one letter to enable variables to be distinguished from constants. The only special character permitted is the underscore "_".

B 2 Overview of Data Types

Data type	Bit	Value range	Explanation	Input formats Examples
REAL	32	±1.175494351E-38 ... ±3.402823466E38	Floating point value IEEE[1] format	0.0, 3.14159, -1.34E-12, -1.2234E-6, 12.6789E10
DINT	32	-2 147 483 648 ... +2 147 483 647	Double integer value with sign	-34355, +23456
INT	16	-32 768...+32 767	Integer value with sign	3, -3, 12345
UDINT	32	0...4 294 967 295	Double integer value without sign	123456787, 4566
UINT	16	0...65 535	Integer value without sign	4000, 66
DWORD	32	0...4 294 967 295 ($0...2^{32}-1$)	Double word	0, 655, 2#0...0...0...0...0...0...0...0001 8#000 000 000 074, 16#0000 0FFF
WORD	16	0...65 535 ($0...2^{16}-1$)	Word	2, 554, 2#0000 0000 0000 0001, 8#000 004, 16#0FFF
BYTE	8	0...255 ($0...2^{8}-1$)	Byte	0, 55, 2#0000 0011, 8#377, 16#0A
BOOL	8	0. 1 (FALSE, TRUE)	Boolean value	0, 1, FALSE, TRUE
DigiTool	32	1970-01-01-00:00:00.000.... 2099-12-31-23:59:59.999	Date+time value	DT#1994-02-14-10:00:00.00
TIME	32	+24d20h31m23s648ms.... -24d20h31m23s648ms	Time value	T#22s T#3m30s T#14m7s

☞ The following applies for the representation of REAL numbers: due to the internal display, only 7 significant positions can be determined during conversion to displayable characters. Very high and very low numerical values are shown as exponents.

[1] IEEE Institute of Electrical and Electronic Engineers; American Association of Experts.

B 2.1 String variables

String variables are used for filing any texts. The variables can be edited e.g. in an FBD program with the **string modules**. Texts thus filed can be used e.g. in the operational printout, the AS criterion window or in free graphics, to describe certain states or provide information.

Data type	Byte	Explanation	Entry formats, Examples
STR8	8	8 character text	FC 1100
STR16	16	16 character text	TIC1234
STR32	32	32 character text	P11400 too low
STR64	64	64 character text	Boilers temp. too high
STR128	128	128 character text	Generator2 speed to high
STR256	256	256 character text	Automation unit malfunctioning

☞ The string variables reserve memory as the other variables in the 32 Kbytes RAM of the resource.

See **Engineering Reference Manual, Functions and Function Blocks, Converter Blocks**.

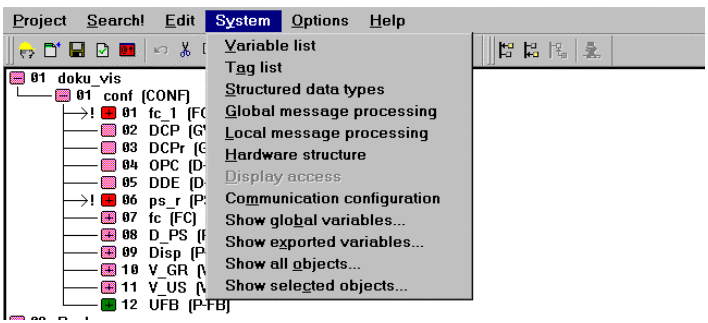
B 3 Variable List

The variable list contains all existing variables in the project. In normal representation it may be wider than the screen display allows for. The other columns can be displayed with the scroll bar below the list. The column width can be adjusted with the mouse, holding the left button down. However a column cannot be made wider than its maximum permissible number of characters. Search criteria can be defined and activated. The total number of entries in the list is shown in the status line as well as the number currently displayed. You can tell from these numbers how many variables satisfy the active criteria, e.g. specific data types or gateway accesses.

B 3.1 Calling up the variable list

→ *System* → *Variable list*

The variable list is called by menu item *System* or the relevant toolbar button.

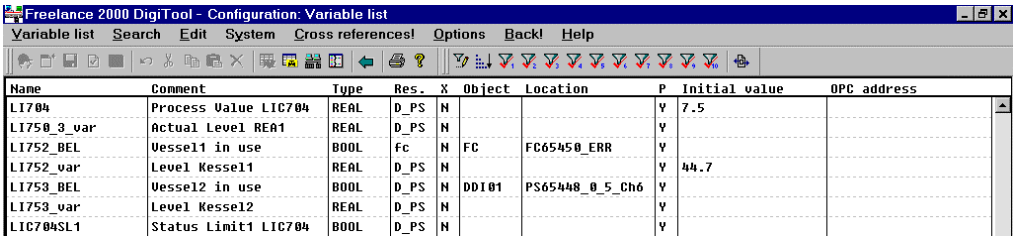


di0310uk.bmp

B 3.2 Structure of variable list

The status line shows the current number of entries thus: *<entries> of <total entries>*. Where search filters are active this enables you to see how many variables meet the search criteria.

The variable list is structured as follows:



Name	Comment	Type	Res.	X	Object	Location	P	Initial value	OPC address
LI704	Process Value LIC704	REAL	D_PS	N			Y	7.5	
LI750_3_var	Actual Level REA1	REAL	D_PS	N			Y		
LI752_BEL	Vessel1 in use	BOOL	Fc	N	FC	FC65450_ERR	Y		
LI752_var	Level Kessel1	REAL	D_PS	N			Y	44.7	
LI753_BEL	Vessel2 in use	BOOL	D_PS	N	DDI01	PS65448_0_5_Ch6	Y		
LI753_var	Level Kessel2	REAL	D_PS	N			Y		
LIC704SL1	Status Limit1 LIC704	BOOL	D_PS	N			Y		

di0313uk.bmp

Name Variable name, max. 16 characters

Comment Comment on variable, (max. 33 characters)

Type Data type, see Page **B-6, Overview of Data Types**

Res. A variable is always allocated to one resource. None of the other resources can read it unless the Export attribute = YES (X) has been given.

X
- Y Variable released for reading by other resources, (Variable input *Export* ☑),
- N Variable available for own resource only, (Variable input *Export* ☐).

☞ An I/O component can only be exported via a variable, never directly; in other words, the I/O component can not be read in other resources by means of the component name. Note that variables which are to be allocated to an I/O component do not feature gateway write rights. See also page **B-28, Station access**

Object, Position For variables assigned to a hardware component the component type and slot or the variable is entered here, e.g. *DDO01* and *PS1_90_5_Ch6* for a channel allocation or *FC* and *FC3_ERR* for a FieldController error signal. Module type (e.g. DDO 01), see **Engineering Manual, System Configuration, Hardware Structure**

Module slot definition, e.g. PS_1_0_2_Ch0

PS_1	Station name in hardware structure
1	Station location
0	Unit (rack)
2	Slot
Ch0	Component name (Channel)

If you double-click in one of these two fields, a dialog appears which enables a hardware component or variable to be selected for allocation.

P

Y Process variables processed from the process image (*Variables via process image* ☑),
N Processing direct from I/O module, (*Variables via process* image ☐).

☞ Changing the P attribute causes only newly-referenced variables to be written via the process image, while existing instances remain unchanged.

Initial value

After the process station has been cold-started, the variable is initialized with this value. See **Page** Fehler! Textmarke nicht definiert.**, Initial values**.

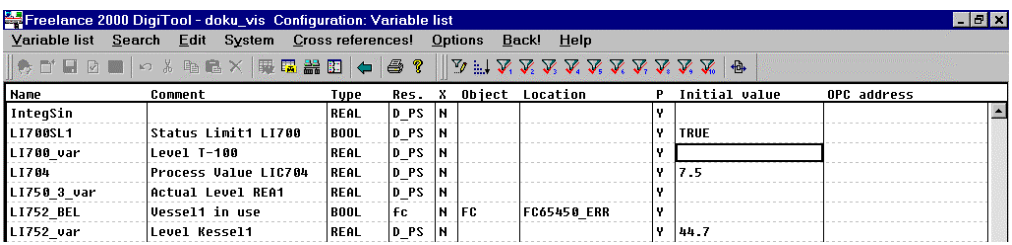
OPC address

Address or name of a variable on the OPC server. For a Freelance OPC gateway this is identical to the variable name in the process station.

☞ **Variables displayed in red** either have no references within the project or they may be system variables. See **Page B-33, System Variables.**

B 3.3 Initial values

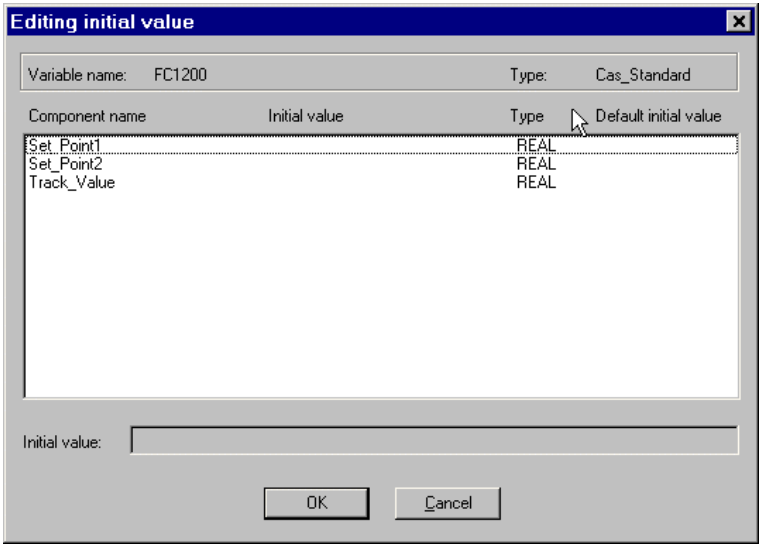
Initial values can be assigned to each variable and to the separate elements of a structured variable; these initial values are adopted following a **cold start** or the **initialization** of a station.
A double-click in the INITIAL VALUE field for a particular variable allows the initial value for that variable to be modified.



Name	Comment	Type	Res.	X	Object	Location	P	Initial value	OPC address
IntegSin		REAL	D_PS	N			Y		
LI700SL1	Status Limit1 LI700	BOOL	D_PS	N			Y	TRUE	
LI700_var	Level T-100	REAL	D_PS	N			Y		
LI704	Process Value LIC704	REAL	D_PS	N			Y	7.5	
LI750_3_var	Actual Level REA1	REAL	D_PS	N			Y		
LI752_BEL	Vessel1 in use	BOOL	fc	N	FC	FC65450_ERR	Y		
LI752_var	Level Kessel1	REAL	D_PS	N			Y	44.7	

di0347uk.bmp

The initial values of the I/O components are entered with the help of the I/O editor.
If the selected variable has a standard data type, then the initial value may be entered directly. In the case of variables with structured data types a mask is displayed which shows all the elements of the structured variable's basic data type.



di0346uk.bmp

By clicking on a variable its default initial value can be replaced by an initial value specifically for that variable. If at least one value has been entered in the mask, this is indicated by -...- in the variable list.

B 3.4 Menu structure variable list

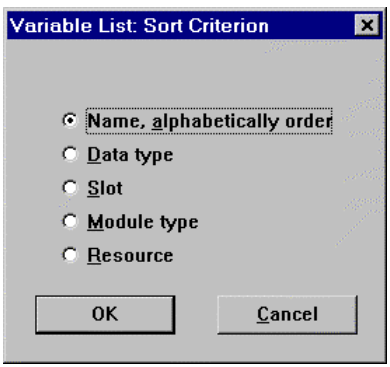
Variable list	*Sort*	
	Normal view	
	Station view	
	Import OPC Variables	*From file*
		Through browser
	Exit	
Search	*Type ahead*	
	Define	
Edit	*Undo*	
	Insert new variable	
	Field	
	Delete field	
	Delete unused variable	
	Delete I/O allocation	
	Cut	
	Copy	
	Paste	
	Delete	
	Import block	
	Export block	
	Station access	
	Assign block to recourses automatically	
	Assign block to recourses manually	
	Assign block to process image	
System	*Structured data types*	
	Tag list	
	Hardware structure	
Cross references!		
Options	*Hardcopy*	
	Color	
	Save column settings	
Back!		
Help	*Contents*	
	Overview	
	Use help	
	About	

B 4 Editing Lists

B 4.1 Sort

→ *Variable list*
→ *Sort* →Select sort criteria

The variable list is output to the screen according to the preselected **sort criterion**.



di0318uk.bmp

B 4.2 Normal view and station view

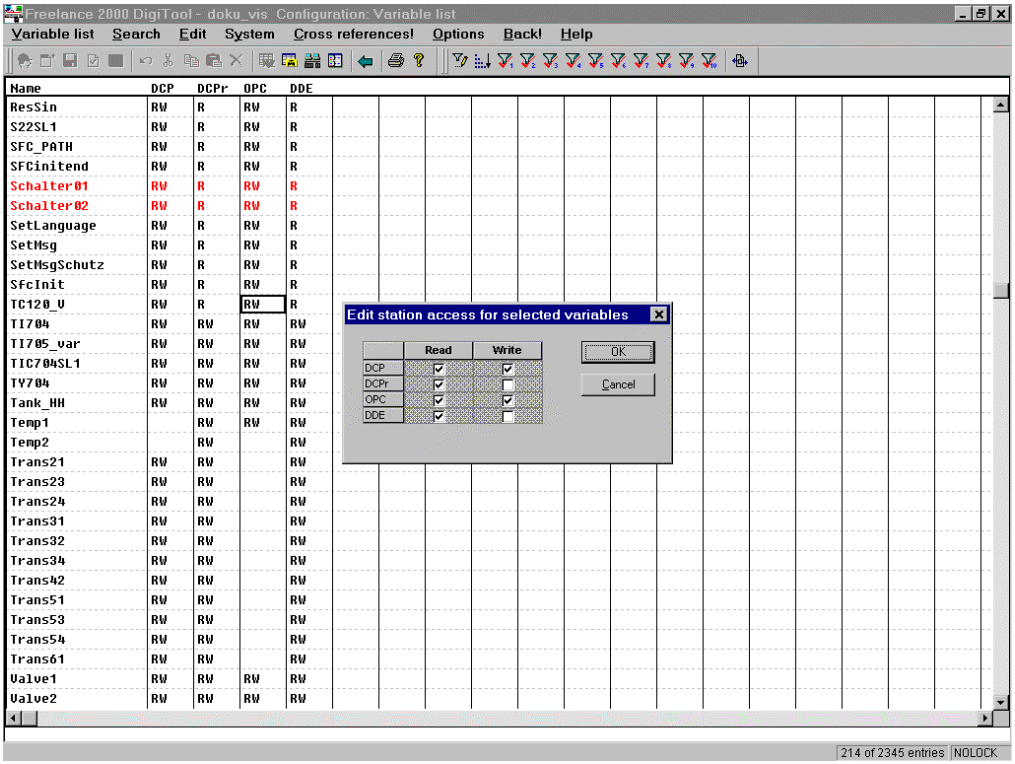
In addition to normal view, a station view may also be selected. In station view parameters are set for each variable to define whether they can be read and/or written via a gateway.

R = Read access - the variable can be read via the gateway.
W = Write access - the variable can be written via the gateway.

→ *Variable list* → *Normal view*

or

→ *Variable list* → *Station view*



di0315uk.bmp

A double-click in the Gateway Resource column or selecting a block with *Edit* → *Station access* enables the READ/WRITE accesses to be modified.

See also the **DigiDDE32** and the **Freelance 2000 - Maestro UX Interfacing** manuals.
See also **page B-28**, **Station access**.

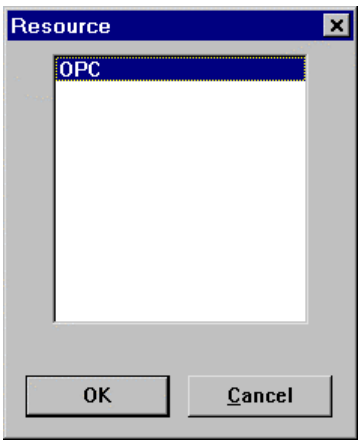
B 4.3 Import OPC variables

→ *Variable list* → *Import OPC variables*

Variables that are to be routed to the Freelance 2000 system via an OPC server are declared to the system using this function. These variables are not assigned to any process station, but remain assigned to the resource of type OPC server.

OPC variables can be displayed on the operator station in free graphic displays and trend displays, in operation logs and signal sequence logs.

First, the external OPC server from which variables are to be used in the project must be selected. The OPC servers configured in the project tree are shown. See also **Engineering Manual, System Configuration, Project Tree, OPC Server**.

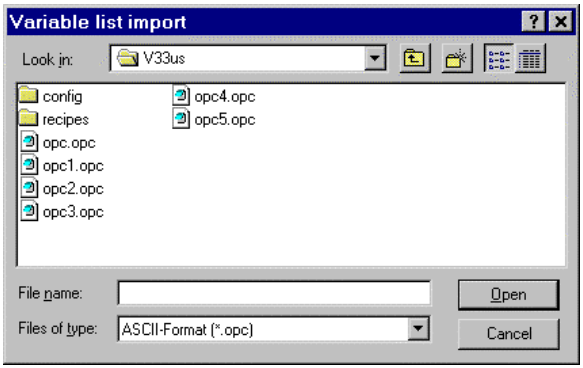


tj001us.bmp

B 4.3.1 Import OPC variables from file

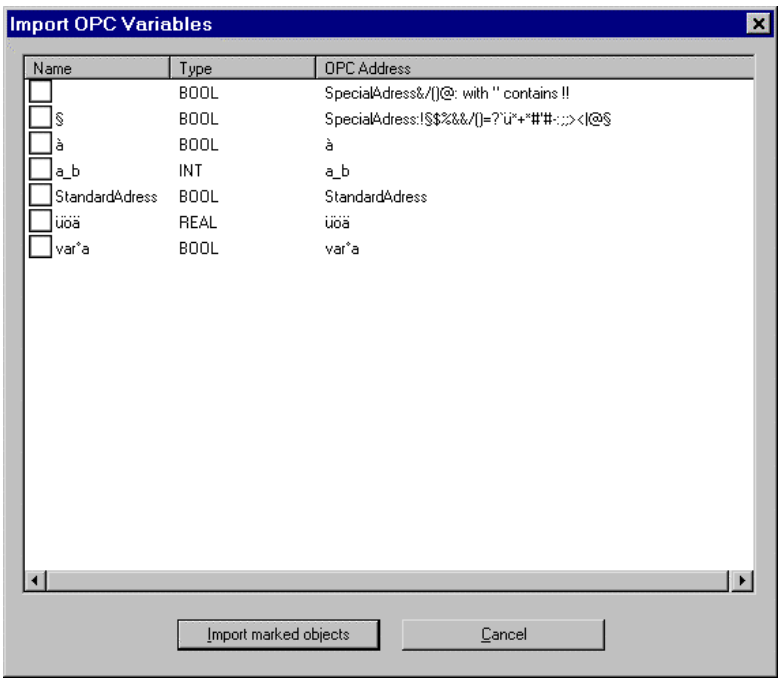
OPC variables can be imported in ASCII form using File-Import.

The import file must be a unicode file.



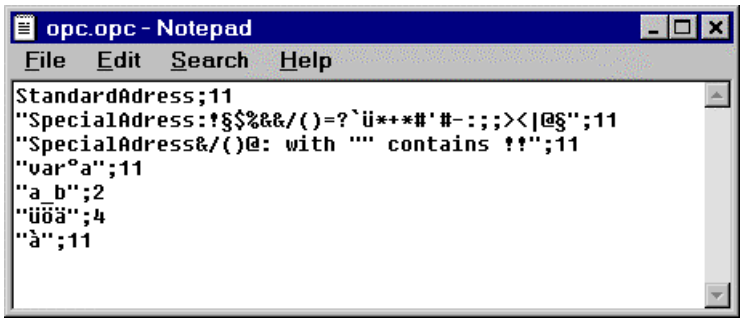
tj003us.bmp

From a list of available variables on the OPC server certain variables can now be marked, and those variables will then be included as variables in the project.



tj002us.bmp

Example:



tj005us.bmp

OPC ADDRESS 1, TYPE NO 1
OPC ADDRESS 2, TYPE NO 2

☞ Semicolons or tab stops may also be used in place of the comma as a separator. However, the same separator must be used consistently within a file.

A variable name is generated automatically from the **OPC address** specified. The name to be used is formed by taking an end string (string starting from the end) from the OPC address; it will be the longest possible end string that is still syntactically correct. OPC addresses containing special characters - in particular the chosen separator and exclamation mark - must be enclosed in quotation marks.
If the OPC address itself is meant to contain a quotation mark, then two quotation marks should be included at that point in the enclosed string.

☞ Variable names may only contain certain special characters such as umlauts and underscore (_).

The **TYPE NO** is the number of the associated data type as defined for the Windows data type "VAR TYPE". If the data type value selected for TYPE NO is not supported by Freelance 2000, that line will be ignored during the importing process.

⚠ Unsigned data types are converted to signed data types. In this way, any values for the target data type that are too large can be represented as negative values!

Data type	TYPE NO signed	TYPE NO unsigned
BOOL	11	
INT	2	18
DINT	3	19
REAL	4	
TIME	7	
BYTE	16	17
STR256	8	

B 4.3.2 Import OPC variables via browse

This procedure is exactly the same as importing OPC variables from file except for selecting a file. The selected variables are imported into the project.

☞ If variables from a Freelance 2000 OPC gateway are to be imported, those variables need to have been assigned read rights at least.

B 4.4 Exit

→ *Variable list* → *Exit*

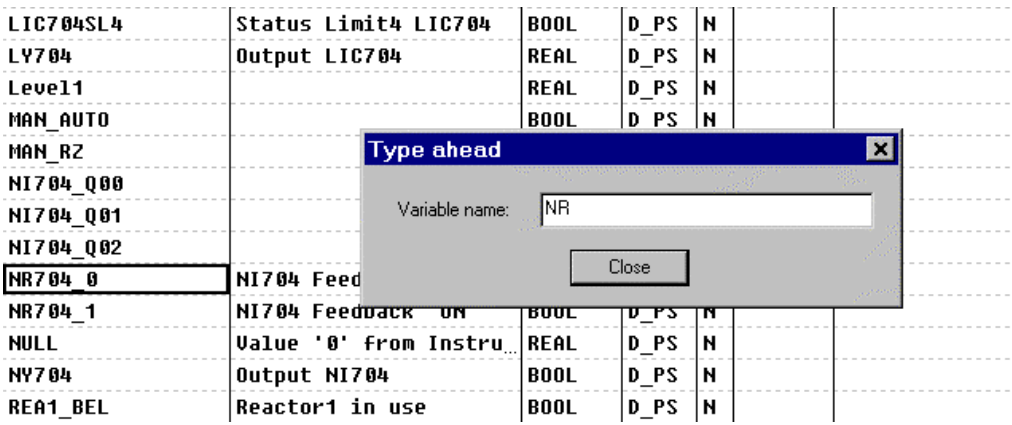
Return to the project tree.

B 4.5 Search

B 4.5.1 Type ahead

→ *Search* → *Type ahead*

The *Type ahead* function enables variables to be searched for by name. When this function is chosen from the menu or shortcut menu a dialog is displayed containing an input field. When a name or the beginning of a name is entered, the list scrolls automatically to the first matching entry.

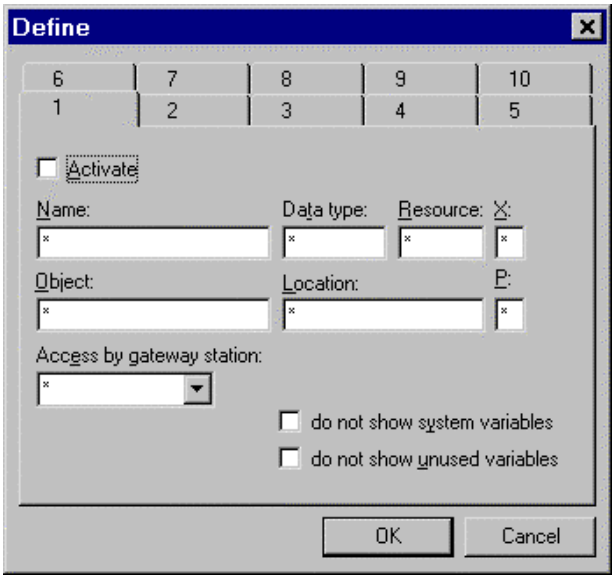


tj0450us.bmp

B 4.5.2 Define search criteria

→ *Search* → *Define* → combine up to 10 search criteria in a dialog.

Entries in the list can be searched and displayed on the monitor according to specified search criteria. For this purpose a dialog is displayed containing 10 identical tabs. These tabs allow search criteria to be defined independently for the 10 different columns in the variable list. Wildcards may be used - * (for several characters) and ? (for any single character).
Each of the 10 search criteria can be activated and deactivated separately on the tab or using the relevant toolbar button.



di0320us.bmp

Activate Activates the search criteria on this tab. After this dialog is closed, all the active search criteria are evaluated and a list is displayed containing entries that satisfy all such criteria.

Access by gateway station
This search criterion is satisfied if read and/or write access has been defined for the variable for the selected gateway.

Do not show system variables
All variables that have been automatically pre-defined by the system can be shown or hidden.

Do not show unused variables
All variables that are defined but not used in a program can be shown or hidden.

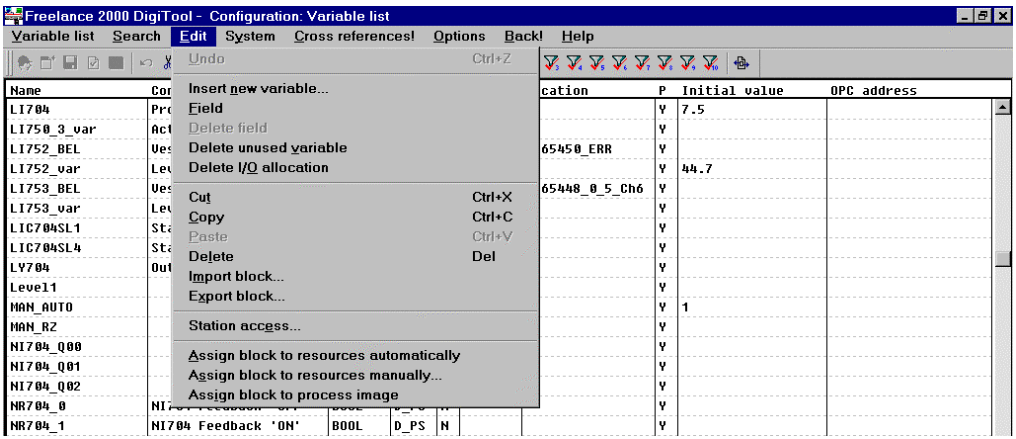
☞ Variables that have had access rights assigned via a gateway but are not used in any program also count as **unused variables**.

The project options in the main menu are used to specify whether or not the **search filter activation** is retained on exiting the variable list. The **search filter configuration** is stored along with the project.
Activated search criteria can be recognized in the status line by the number of entries displayed and by the corresponding toolbar buttons that are pressed. The configured search criteria are indicated in the form of tool tips on the toolbar buttons.

B 4.6 Edit list entries

→ *Edit*

A number of functions are provided for editing the individual list entries. For example, *Undo* can be used to undo the last action, new entries can be inserted, entries can be deleted, cut or copied. Blocks or variables can also be imported and exported.



di0338uk.bmp

B 4.6.1 Undo

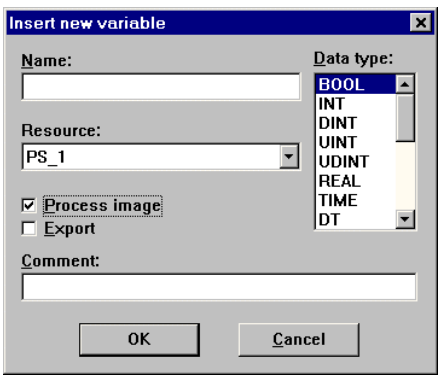
→ *Edit* → *Undo*.

The last change is undone and the old status restored. If *Undo* cannot be performed, the menu item cannot be chosen (displayed in gray).

B 4.6.2 Insert new variable in list

→ *Edit* → *Insert new variable*.

After the menu item *Insert new variable* has been chosen, a window is displayed. The parameters for the variable must be entered in this window.



di0335uk.bmp

Name	Enter variable name, max. 16 characters.
Data type	Select data type from a list of data types.
Resource	Enter the resource by means of a selection list.
Variable via Process image	☑ The variable is read via the process image, ☑ The variable is read not via the process image, but directly at the time of processing. This results in a greater load on the CPU module!
Export	☑ The variable can be read in other resources. ☐ The variable can only be read or written by its own resource.

☞ An I/O component cannot be exported directly, but only with the assistance of a variable: this means that the I/O component cannot be read by other resources through the component name. It is also important to remember that variables which are to be assigned to an I/O component do not have the write rights of a gateway.

Comment — Comment in the form of free text.

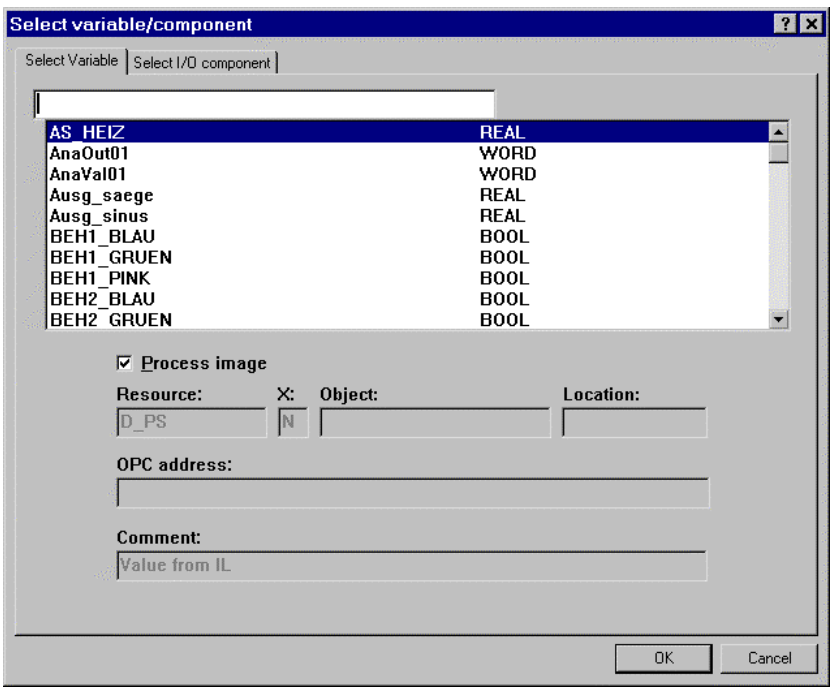
☞ When Search is activated, i.e. the list is not displayed in its entirety, it is not possible to insert any new variables.

B 4.6.3 Create new variable in program

It is possible to define new variables directly in the program editors. Variables that are to be used in a program but have not yet been declared in the project can be inserted directly in the program. Once a new name has been entered, the dialog described in the previous section for declaring a variable is displayed automatically.

B 4.6.4 Insert existing variable in a program

At every point at which a variable needs to be defined in a program the function key **F2** can be pressed. The following dialog appears:



di0345uk.bmp

The variable to be inserted in the program can be selected.

Variable via Process image — This parameter can be set to define whether or not the value of the variable can be read from the process image. See **Engineering Manual, System Configuration, Project Tree, Process Image**.

The other details, e.g. resource, are shown for information purposes and can only be modified in the variable list itself.

B 4.7 Change variable entries

If existing variables are modified, this can affect the different programs. In order to avoid errors a list of the affected programs is displayed when changes are made. A decision can then be made as to whether or not the changes are to be carried out.

 Select required field by double-clicking

Depending on the field selected, the new value can either be entered directly or modified by means of a dialog.

B 4.8 Edit a field in the list

 → Select required field by double-clicking (the selected field is emphasized with a border). Cursor is positioned at the last entry position
→ Cursor-click on entry position within the field,
→ Enter changes.

The text content of the selected field can be modified. After the change has been entered, another window will be displayed if appropriate to query whether this change should be applied throughout the entire project or only in certain programs.

B 4.9 Delete field

 Certain entries in fields cannot be explicitly deleted using this command. In the case of the **variable list** the fields Name and Type fall into this category.

If a whole line in the list is selected, then the variables may be deleted.

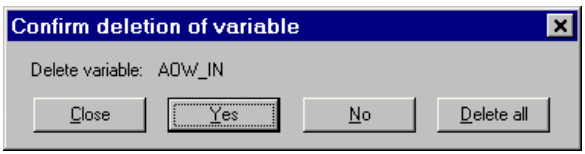
 → Click on required field (highlighted by a border, cursor positioned at last entry position)
→ DELETE button.

The text parts of a list entry can be deleted directly with the cursor. This is achieved by clicking on the field, positioning the cursor at the beginning of the section to be deleted, selecting the area for deletion by holding down the left mouse button, and lastly removing the text thus selected by pressing the DELETE button.

B 4.10 Delete unused variables

☞ All entries with no cross-references (these variables are identified by a red color) are deleted following a query for confirmation. The system variables cannot be deleted.

→ *Edit* → *Delete unused variables*



di0340uk.bmp

YES	The variable that is displayed is deleted.
DELETE ALL	All unused variables (all variables in red) are deleted.
NO	The variable that is displayed is not deleted, and the next variable is displayed.
CANCEL	Quits the query mask.

☞ Variables that have had access rights assigned via a gateway but are not used in any program also count as **unused variables**.

B 4.11 Block processing

Only one block can be defined respectively. It comprises a number of whole lines from the list that are selected; the block can be selected as follows:

→ Cursor click where the block is required to start,
→ Drag the mouse to the end of the block area with the left button held down.

The resulting block is identified and is also retained when the left mouse button or the SHIFT key is released.

B 4.11.1 Cut

 → Select block → *Edit* → *Cut*

A defined block is removed from the text section and stored in the clipboard.
The command *Paste* is used to insert this stored block in any other position.

B 4.11.2 Copy

 → Select block → *Edit* → *Copy*

A defined block is copied and stored in the clipboard.
The command *Paste* is used to insert this block in any other position.

B 4.11.3 Paste

 → Select block → *Edit* → *Paste*

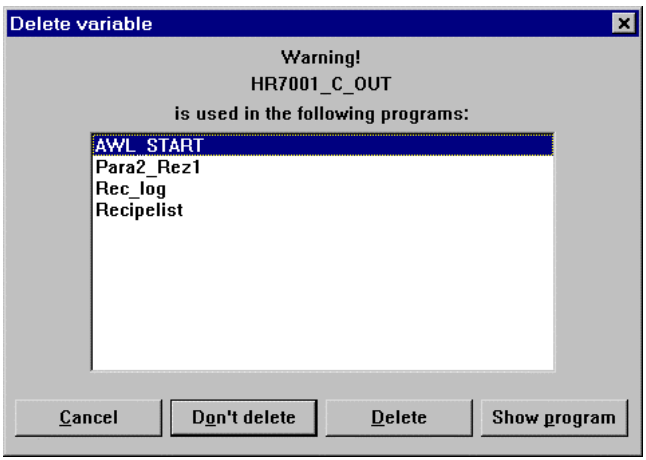
A copied or cut block in the clipboard is inserted at the position defined by the cursor.

 Since the variable names must be changed the same window is displayed as for the menu item *Insert new variable*.

B 4.11.4 Delete

→ Select block → *Edit* → *Delete*

After a query for confirmation a defined block is removed from the text within a window.



di0348uk.bmp

DON'T DELETE	Selected variable is not deleted
DELETE	Selected variable is deleted
SHOW PROGRAM	Jump to the selected program.
CANCEL	Return to the variable list

B 4.11.5 Import

→ Select block → *Edit* → *Import*

A file that has been stored using *File export* is read in from a data medium (hard disk, floppy). Another window is displayed to enable the path and file name to be entered.

If variable names are discovered during the importing process which already exist (same names) in the project, the first instance normally has the suffix ....00 added, the second instance has ....01 added etc..

B 4.11.6 Export

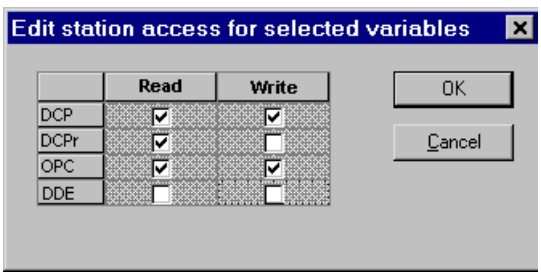
→ Select block → *Edit* → *Export*

A defined block is stored as a file on a data medium (hard disk, floppy). Another window is displayed to enable the path and file name to be entered. This file can be copied across project boundaries into other projects using the *File import* command.

When variables are exported, the export attribute X = Yes is lost since the resource needs to be reallocated after an import.

B 4.12 Station access

→ Select block → *Edit* → *Station access*



di0344uk.bmp

If the variable is to be read or written through a gateway station this access must be enabled
- in the project tree on the gateway station and
- in the variable list

Variables which are to be assigned to an I/O component must not have the write rights of a gateway.

See also **DigiDDE32**, **OPC for Freelance 2000** and **Freelance 2000 - Maestro UX Interfacing** manuals.

See also **Page B-13, Normal view and station view.**

B 4.12.1 Assign block to resource automatically

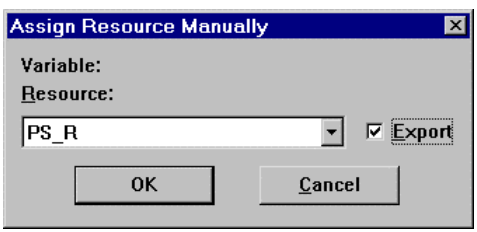
Following a block import none of the variables that have been newly added to the project database during the import process have yet been allocated to a resource. Variables which already existed in the project retain their resource allocation. If automatic allocation has been selected, variables that have been selected by block selection within the variable list are assigned automatically to resources according to the programs that the variables are referencing. The ASSIGN button can be used to subsequently manually assign those variables which proved impossible to assign automatically. See **Page B-29, Assign block to resource manually**.

→ Select variable or block,
→ *Edit* → *Assign block to resource automatically*,
→ the resource is assigned and entered in the *Res* column.

If the variable is not yet used in the project under this name, no resource (process station) can be assigned automatically.

B 4.12.2 Assign block to resource manually

→ Select variable or block,
→ *Edit* → *Assign block to resource manually*



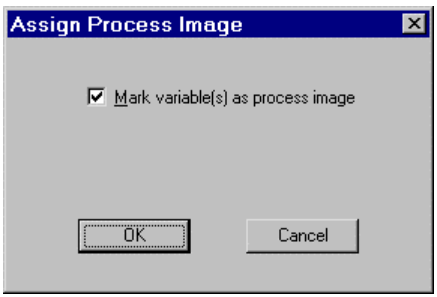
di0352uk.bmp

Each variable should be assigned to precisely one resource (process station). Following a block import none of the variables that have been newly added to the project database during the import process have yet been allocated to a resource. Variables which already existed in the project retain their resource allocation. Manual resource assignment can be used to select one of the existing process stations in the project. All the variables selected in the block are then assigned to this resource and none other.

If *Export* is called, variables from other resources can be read in.

B 4.12.3 Assign block to process image

→ Select variable or block,
→ *Edit* → Assign block to process image



tj004us.bmp

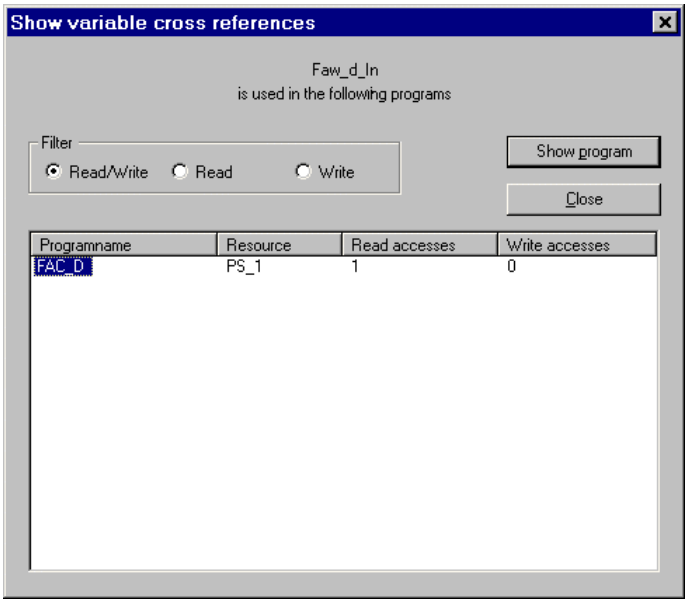
All the variables selected in a block are assigned to a task through the process image. See also **Engineering Manual, System Configuration**, **Project Tree, Process Image**.

B 4.13 Cross-references

All cross-references for a variable can be shown in a list through *Cross-references*. Cross-references are references to this variable in programs, displays, logs etc., in other words to places in which this variable is used.

→ Select field → *Cross-references* or F5 button,

A window displays the names of affected programs.



di0339uk.bmp

SHOW PROGRAM — Calls the program and pre-selects this variable or calls the module to whose I/O component this variable is assigned.

SHOW DECLARATION — Jumps to a corresponding I/O component in the I/O editor if there is a variable assigned to an I/O component. Otherwise the variable list remains selected.

Filter — A filter enables just those variables to be displayed that are being edited using the appropriate programs in read (sink) or write (source) mode.

B 4.14 Hardcopy

 → *Options* → *Hardcopy*

The contents of the screen are output to the printer

B 4.15 Adjust colors

 → *Options* → *Colors*

The variable list offers the facility of a free choice in defining the colors of unused variables.

B 4.16 Store column width

 → *Options* → *Store column width*

The column width setting is saved.

B 4.17 Back

 → *Back*

Returns to the display called previously.

B 5 System Variables

When a new resource is created, certain system variables are automatically declared for the resource and are made available to the user.
These variables are global, i.e. they can be read by other resources throughout the entire system.

They are recorded in the variable list and may be freely accessed or edited from within the project. Thus a program may be started or information generated when a defined CPU load is exceeded. The first four characters of the name structure show the resource name, followed by the assigned variable name, e.g. DPS1.StationNo.

The system variables, with the exception of the variables for lateral communication, are not shown in the list of **global variables in the resource**. The reason behind this is that these variables are stored elsewhere in the system.

The key shown below relates to the following explanation of system variables and their significance:

xxxx = name of the resource;
column **P**: X = system variable of a resource D-PS, D-FC or D-PS/RED
column **G**: X = system variable of a gateway station D-GS or D-GS/RED

In general version numbers are coded as three variables: xMajorVerNo, xMinorVerNo, and xPatchVerNo.

B 5.1 System variables with project information

Variable name	Data type	P	G	Designation
xxxx.ProjectName	STRING16	X	X	Name of current project.
xxxx.CMajorVerNo	UINT	X	X	Current major project version number
xxxx.CMinorVerNo	UINT	X	X	Current minor project version number. It increases each time a program is loaded or deleted.
xxxx.CPatchVerNo	UINT	X	X	Current version number for project "amendments". It increases every time the function block is changed

B 5.2 System variables with resource information

Variable name	Data type	P	G	Designation
xxxx.StationNo	UINT	x	x	Station number of the resource
xxxx.StationType	UINT	x	x	Station type of the resource 4 = D-PS or D-PS/RED or D-FC 5 = D-GS or D-GS/RED
xxxx.MaxObjNo	UINT	X	X	Maximum number of objects which may be handled by the resource
xxxx.GlobVarSize	UINT	X	X	Size of RAM for global variables
xxxx.PRAM_Size	UDINT	X	X	Size of write-protected RAM in bytes (RAM for user configuration)
xxxx.PRAM_Free	UDINT	X	X	Free write-protected RAM currently available in bytes (configuration memory)
xxxx.RAM_Size	UDINT	X	X	Size of RAM in bytes (working memory)
xxxx.RAM_Free	UDINT	X	X	Free RAM currently available in working memory
xxxx.CPU_Load	UINT	X	X	Current CPU load (%)
xxxx.DateTime	DT	X	X	Current date and time at resource (Local time)
xxxx.UserStopped	BOOL	X		Boolean variable, logic = 1 when the station is shutdown from DigiTool
xxxx.MsrStopped	BOOL	X		Boolean variable, logic = 1 when the station is shutdown via a RUN/STOP switch on the CPU module
xxxx.ResState	UINT	X		Displays current state of the resource. 1 = no operating system 2 = cold start 4 = cold start stopped 8 = running 16 = stopped 32 = warm start 64 = warm start stopped 128 = standby 256 = starting 512 = stopping
xxxx.OMajorVerNo	UNIT	X	X	Part 1 of the operating system version number
xxxx.OMinorVerNo	UNIT	X	X	Part 2 of the operating system version number
xxxx.OPatchVerNo	UINT	X	X	Part 3 of the operating system version number

Variable name	Data type	P	G	Designation
xxxx.Configuring	BOOL	X		Boolean variable, logic = 1 when the station is being configured by DigiTool
xxxx.EMajorVerNo	UINT	X	X	Current major EPROM version number
xxxx.EMinorVerNo	UINT	X	X	Current minor EPROM version number
CPURack	UINT	X		ID of the rack which the currently active CPU module (Primary CPU) is plugged into.
CPUSlot	UINT	X		Slot holding the currently active CPU module (Primary CPU).
RadioClkAv	BOOL	X		Boolean variable set to logical 1 if the process station is synchronized by a radio clock. The radio clock does not need to be connected directly to the process station. The synchronization can also be performed by another process station which has a radio clock connected.
xxx.TSynchInst	BOOL		X	Boolean variable set to logical 1 if the gateway sends time synchronization messages to external systems. This functionality can be activated either by configuration in DigiTool (enable external time synch.) or by loading the gateway from Maestro UX (gateway configured as time master for the workstations).

B 5.3 System variables with information of a redundant resource

Variable name	Data type	P	G	Designation
xxxx.MainCPUPrim	BOOL	X		Boolean variable set to logical 1 when the CPU module in the central unit (slot with rack-ID = 0 and slot ID = 0) is active (Primary CPU). This variable is set to logical 0 when this CPU module is passive (Secondary CPU).
xxxx.RedCPURack	UINT	X		ID of the rack which the passive CPU module (Secondary CPU) is plugged into. In the event of a redundancy toggle the status changes from RedCPURack and MainCPUPrim.
xxxx.RedCPUSlot	UINT	X		Slot-ID of the passive CPU module (Secondary CPU).
xxxx.RedState	UINT	X	X	Redundancy status 0 = no redundancy 1 = no secondary 2 = not sync 3 = sync 128 = Redundancy error
xxxx.RedLinkLoad	UINT	X		Load on the redundancy link.
xxxx.StationLoad	UINT	X		Load on the station (combination of CPU_Load and RedLinkLoad).
xxxx.RedBufLow	UDINT	X		Remaining storage space for redundancy data.

B 5.4 System variables for powerfail on voltage failure

Variable name	Data type	P	G	Designation
xxxx.NoPowerFail	UINT	X		Present number of PowerFails which did not lead to a warm start. The variable is initialized at zero after a cold start
xxxx.PowerOffTim	TIME	X		Length of last power failure which led to a warm start. It is counted from the time the power failure occurred to the restarting of the operating system.

B 5.5 System variables for error handling task

Variable name	Data type	P	G	Designation
xxxx.ErrorNo	UDINT	X		Error number of last error which rendered a task "unrunnable".
xxxx.ErrorProgram	UINT	X		Variable shows the object number of the program which triggered the last error in the process station.
xxxx.ErrorTask	UINT	X		Variable shows the object number of the task which triggered the last error in the process station.

B 5.6 System variables for I/O Communication

y denotes the rack ID (numbered consecutively from 0 to 4) and **z** the module slot (numbered consecutively from 1 to 8), e.g. DPS1.IOBootT-1-3.

Variable name	Data type	P	G	Designation
xxxx.IOBootT-y-z	BOOL	X		State of I/O module, logic = 1 when an I/O module is identified.
xxxx.IOBoard-y-z	UINT	X		Type of I/O module. The following modules are defined

10	DDI 01, 32 x 24 V DC	56	DCP 10, gateway
11	DDI 04, 28 x Namur initiators or 12 x 3/4-wire initiators	60	DDO 02, 16 x 230 V AC/DC
12	DDI05, 32 x 120/230 V AC	61	DDI 02, 16 x 24..60 V AC/DC
20	DDO 01, 32 x 24 V DC, 0,5 mA	62	DDI 03, 16 x 90..230 V AC
30	DAI 01, 16 x 0/4..20 mA, 50 Ohm	63	DDO 03, 16 x 24..60 V AC/DC, read back
31	DAI 02, 16 x 0..10 V DC	64	DDO 04, 16 x 115..230 V AC, read back
32	DAI 03, 16 x 0/4..20 mA, 250 Ohm	70	DAI 04, 8 x PT100/mV
35	DAI 05, 16 x 0/4..20 mA, MU powering	80	DFI 01, 4 x f <= 45 kHz
40	DAO 01, 16 x 0/4..20 mA	89	DLM 01 - Link module
50	DCP 02, CPU	90	DLM 02 - Link module
51	DCP 10, CPU	100	DCO 01, 4 x RS 485/422/232 C
52	DCP 02, gateway		

xxxx.IOForce-y-z	BOOL	X		Shows forcing state of channel on the I/O module. Boolean variable, logic = 1 when a channel is forced on the module.

B 5.7 System variables with information for lateral communication

Variable name	Data type	P	G	Designation
xxxx.SendErr	BOOL	X		Logical 1 if resource xxxx cannot transmit
xxxx.yyyy.RcvErr	BOOL	X		Logical 1 if resource xxx has not received any values from resource yyyy within twice the transmission cycle time of resource yyyy. An alarm is also given in this case if values have already been received once from resource yyyy. When values are received the RcvError is automatically reset to logical 0.

☞ The variables xxxx.yyyy.RcvErr are generated automatically if export flags are set by variables for lateral communication.

B 6 Structured Data Types

Application-specific data types can be created, i.e. defined in addition to the structured ones, with the aid of the editor. These user-defined data types are included in the data type selection list and can be selected like standard ones. In this way a series of data (max. 256) can be transmitted via a structured variable. For example, all the important control signals can be switched to another station by using **one** variable instead of transmitting all the structured data types separately.

B 6.1 Calling up structured data types

→ *System* → *Structured data types*

B 6.2 Define a new data type

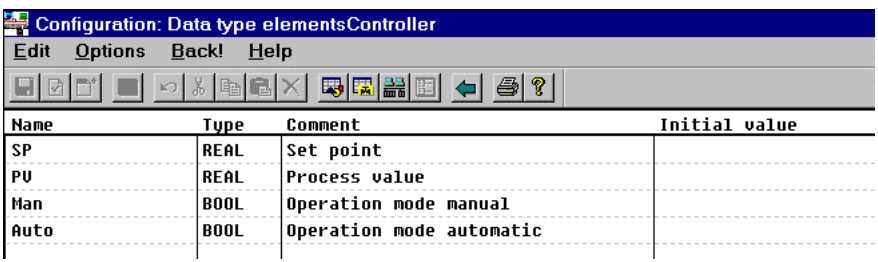
Insert a new data type name into the list of structured data types, confirm with OK.

→ *Edit* → *Insert a new data type*

B 6.3 Creating data type components

The components of the new defined data type can be entered with:

→ *Data Type*



di0308uk.bmp

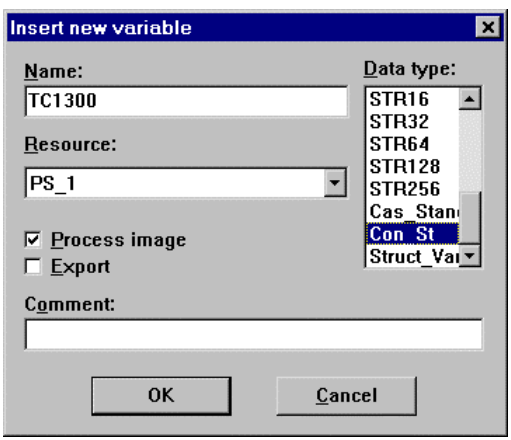
See also page **B-6, Overview of Data Types**

The structured data types which are to be available under the new data type are then entered.

Name	Name of structured data type (max. 16 characters)
Type	Data type as BOOL or REAL. See also **Page B-6, Overview of Data Types**.
Comment	Comment
Initial value	A default initial value may be entered. When a warm start is made, this value is used as the structured data type for all variables which have that data type. See also **Page** Fehler! Textmarke nicht definiert.**, Initial values**.

B 6.4 Insert a new variable with structured data type

→ *System* → *Variable list* → *Edit* → *Insert a new Variable*



di0307uk.bmp

Corresponding variables may be adopted using the new **Control** data type. For example, several controls of the same type may be provided with variables just by opening a variable, e.g. **TC120_V**, with the new **Control** data type. All components with their basic data types are then available for this structured variable.

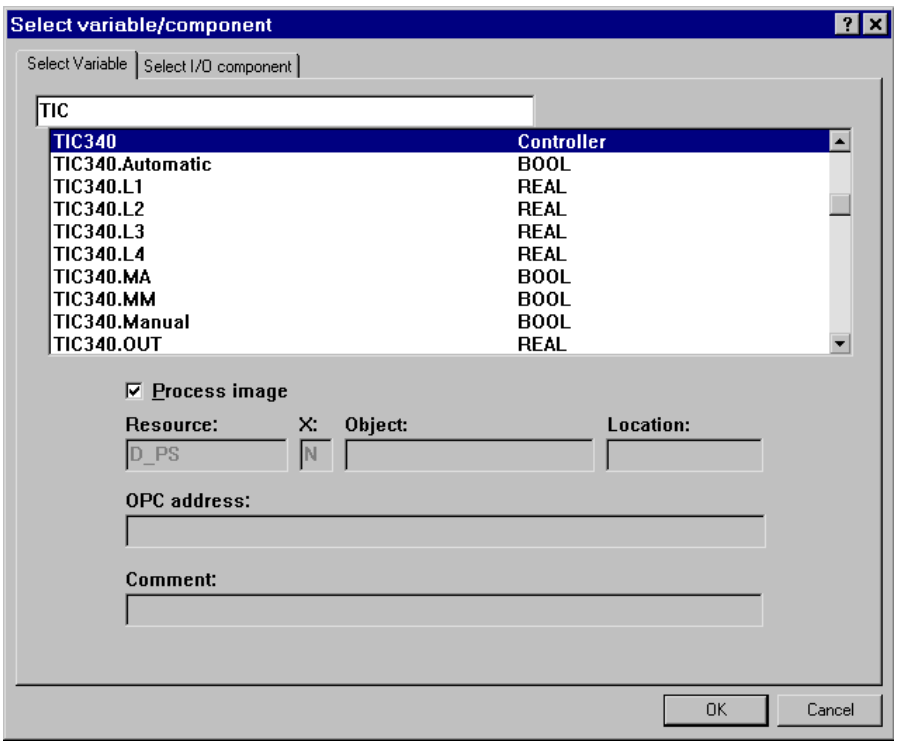
In the example below the new variable TC120_V is assigned the structured data type **Control**. The following components of variable **TC120_V** are thus available:

TC120_V.X	REAL	Track value
TC120_V.W	REAL	Set point
TC120_V0.Y	REAL	Output value
TC120_V.MM	BOOL	Manual
TC120_V.MA	BOOL	Automatic etc.

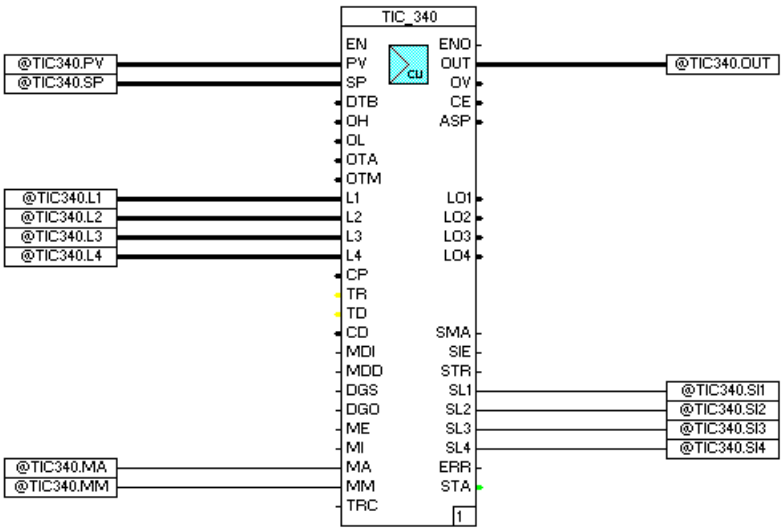
B 6.5 Using a structured data type in a program

→ e.g. select a read or write variable in a function block diagram.

→F2



di0305uk.bmp



di0349uk.bmp

In the window shown above various components of the structured variable TC120 (data type control) have been used.

C Tags

Name	R	Short text	Long text	Type name	L	P
AI_TR	+		Analog input transfo...	AI_TR	S	@
AI_TRT	?		Analog input transf....	AI_TRT	S	@
AI_TR_T	+		Analog input transfo...	AI_TRT	S	@
AO_TR	?		Analog output transf...	AO_TR	S	@
BA700	+	Auto	Automatic0 state	M_BOUT	S	@
BA700_1	+			MONO_F	S	@
BA700_2	+			MONO_F	S	@
BIT1	+	LSB		TR	S	@
BIT2	+			TR	S	@
BIT3	+	MSB		TR	S	@
CSTBO	?		BOOL constant	CSTBO	S	@
CSTBY	?		BYTE constant	CSTBY	S	@
CSTDI	?		DINT constant	CSTDI	S	@
CSTDT	+		DT constant	CSTDT	S	@

Contents

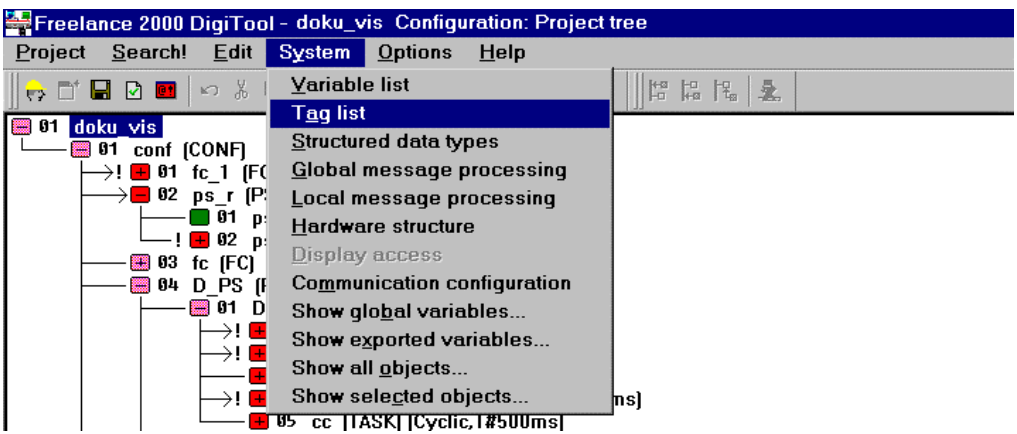
C 1 General Description - Tag List

All function blocks (tags) configured in a project as well as the modules configured in the hardware structure are organized by the system and made available to the user in the tag list.
These lists are automatically generated or updated when a project is configured. Existing data may be output to data media or imported from these media.
Data files are in ASCII text format with CSV (**c**omma **s**eparated **v**alues).
It is possible to use 16 character long tag names (KKS) instead of 12 character long tag names. To be able to use 16 character long tag names this option must be ordered separately. See also **Engineering Manual, System Configuration, Project Manager, Expanding tag names.**

Search criteria can be defined and activated. The total number of search criteria is displayed in the status line along with the number of entries currently displayed in the list. It is thus possible to see how many tags satisfy the active search criteria, e.g. specific block types or gateway accesses.

C 2 Calling Tag List

→ *System* → *Tag list*



di0312uk.bmp

C 3 Structure of Tag List

The tag list contains all the function blocks specified in the project.
It is structured as follows:

Tag list Search Edit System Cross references! Options Back! Help

Name	T	A	R	Short text	Long text	Type name	L	P
DOS_A_us	S	E	+	Dosing		DOS_A	S	@
FIC1226us	S	E	+	Flow 26		C_CR	S	@
FIC1512us	S	E	+	Flow 12		C_CU	S	@
FX1226us	S	E	+	Motor		IDF_2	S	@
H700us	S	E	+	Inlet	Inlet valve	IDF_1	S	@
IDF_T_us	S	E	?	Short text		IDF_T	U	@
L704_us	S	E	+			DISLOG	S	@
LI328us	S	E	?	Scale dosing		DOS_S	U	@
LI700us	S	E	+	Level	Level T-100	CT_ANA	S	@
LIC704us	S	E	+	Level R-100	Controller reactor	C_CS	S	@
NI704us	S	E	+	Motor	Reactor motor R-100	IDF_1	S	@
R704_us	S	E	+	Trend		TREND	S	@

di0314uk.bmp

Name — Tag name of function block, max. 12 characters

☞ It is possible to use 16 character long tag names (KKS) instead of 12 character long tag names. To be able to use 16 character long tag names this option must be ordered separately. See also **Engineering Manual, System Configuration, Project Manager, Expanding tag names.**

T — Object type of entry:

S Standard name.
Name of a function block, name of an SFC program, name of a module or a hardware structure object. Other than these, all unused tags or objects are labeled with 'S'.

F Formal name.
Entries with which function blocks are addressed within the class definition of a user-defined function block are labeled with 'F'.

T Template name.
All template entries in the hardware structure are labeled with 'T'.

A — Plant area of tag, max. 15 plant areas (A...O) possible.

☞ The plant area assignment is preserved in project export and import, but not in block export and import.

R — State of processing, only information,
+ function is in processing (Processing ☑),
- function is not in processing (Processing ☐),
? Processing not defined (Processing ▩),

☞ For user function blocks, sequential function chart programs and I/O modules the state of processing is displayed with "?".

Short text — Short text for tag, max. 12 characters

Long text — Long text for tag, max. 30 characters

Type name — Abbreviated text for function block type, e.g. M_ANA for analog monitoring.
Changes may be made via a selection window listing the relevant function block types. See also **Engineering Reference Manual, Functions and Function Blocks.**

L — Library type
S standard library type
U user function blocks
E extra library type (SFC program)

P — # function block isn't checked
@ function block is checked

☞ Entry P cannot be changed within the tag list!
See also **Engineering Manual, System Configuration, Project Tree, Plausibility check.**

C 3.1 Menu Tag list

Tag list	*Sort*	***System***	*Tag list*
	Normal view		*Structured data types*
	Station view		*Hardware structure*
	Exit		
		Cross references!	
Search	*Type ahead*		
	Define	***Options***	*Hardcopy*
			Color
Edit	*Undo*		*Save column settings*
	Insert new tag		
	Edit field	***Back!***	
	Delete field		
	Delete unused tags	***Help***	*Contents*
	Cut		*Overview*
	Copy		*Use help*
	Paste		*About*
	Delete		
	Export block		
	Import		
	Station access		
	Area		
	Tag type assignment		

C 3.2 Changing tag list settings

Changing existing tags may affect some programs. The relevant programs are listed when changes are made as a precaution. These lists may be used to decide whether the changes should take effect.

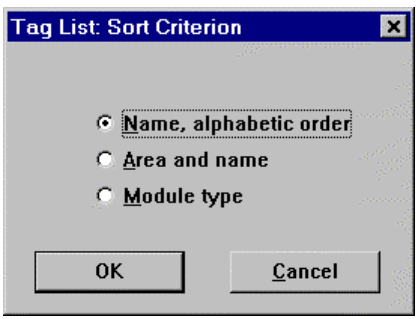
→ Select desired field with double click
→ Carry out or abandon changes.

C 4 Editing the Tag List

C 4.1 Sort

Tag list → *Sort* → Select sort criterion

The variable list or tag list entries are output to the screen according to the preselected sort criterion.



di0317uk.bmp

Name, alphabetic order	Sorted alphabetically according to name
Area and name	Sorted according to plant area
Module type	Sorted according to module type

OK	Sorting is activated.
CANCEL	Sort action is aborted.

C 4.2 Normal view and station view

As well as the normal view, a station view can also be selected, in which the READ (R) and WRITE (W) accesses for the gateway stations can be configured. Access = X can be specified for the individual operator stations, allowing this tag to be available in the tag list for the relevant operator station

→ *Tag list* → *Station view*

Tag list Search Edit System Cross references! Options Back! Help

Name	V GR	V US	DCP	DCPr	OPC	DDE
H700gr	X	X	RW	RW	RW	RW
H700us	X	X	RW	RW	RW	RW
HS704	X	X	RW	RW	RW	RW
HWSYS	X	X	RW	RW	RW	RW
IDF_T_gr	X	X	RW	RW	RW	RW
IDF_T_us	X	X	RW	RW	RW	RW
L704_gr	X	X	RW	RW	RW	RW
L704_us	X	X	RW	RW	RW	RW
LI328gr	X	X	RW	RW	RW	RW
LI328us	X	X	RW	RW	RW	RW
LI700_1	X		RW	RW	RW	RW
LI700gr	X		RW		RW	R
LI700us	X		RW		RW	R
LI720	X		RW		RW	R
LI720_1	X		RW		RW	R
LIC704gr	X		RW	RW	RW	RW
LIC704us	X		RW	RW	RW	RW
L_704	X	X	RW	RW	RW	RW

di0316uk.bmp

Change the READ/WRITE access with a double click in the resource column, or with a block selection with *Edit* → *Station access*.

Also refer to page **C-20, Station access,** to the manual **DigiDDE32** and to the manual **Coupling Freelance 2000 - Maestro UX**.

C 4.3 Exit

Tag list → *Exit*

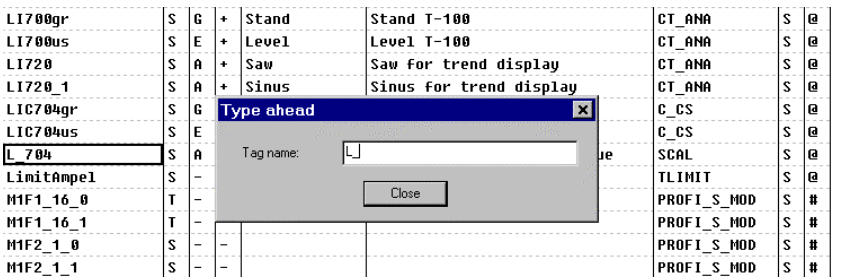
Back to project tree structure.

C 4.4 Search

C 4.4.1 Type ahead

→ *Search* → *Type ahead*

The *Search ahead* function enables you to search for tags by name. After this function has been chosen from the menu or the shortcut menu, a dialog appears containing an input field. When a name or the beginning of a name is entered here the list scrolls automatically to the first matching entry.

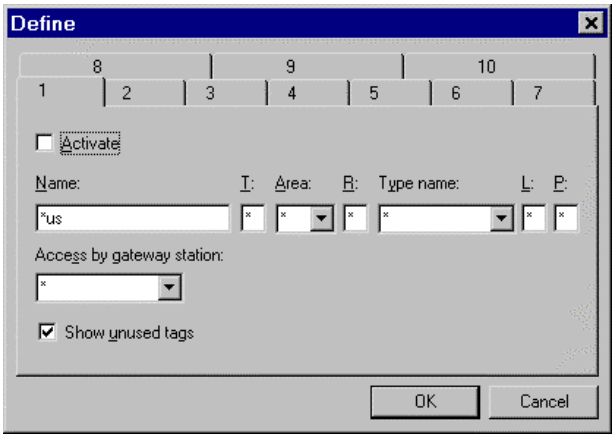


tk0440us.bmp

C 4.4.2 Define search criteria

→ *Search* → *Define* → Compile up to 10 search criteria in a dialog

A search can be performed for entries in the list that conform to the specified search criteria, and these can then be displayed on the screen. For this purpose, a dialog is displayed with 10 identical tabs. The use of wildcards is permitted, such as * (for more than one character) and ? (for any single character).
Each of the 10 search criteria can be activated and deactivated separately on the tab or by means of an appropriate toolbar button.



tk0441us.bmp

Activate Activate search criteria on this tab. After this dialog is closed all the active search criteria are evaluated, and a list is displayed containing entries for which all the criteria are satisfied.

Access by gateway station
The search criterion is satisfied if read or write authorization has been defined for the tag and for the selected gateway.

Show unused tags
All those tags that have been defined but not used in a program can be either shown or hidden.

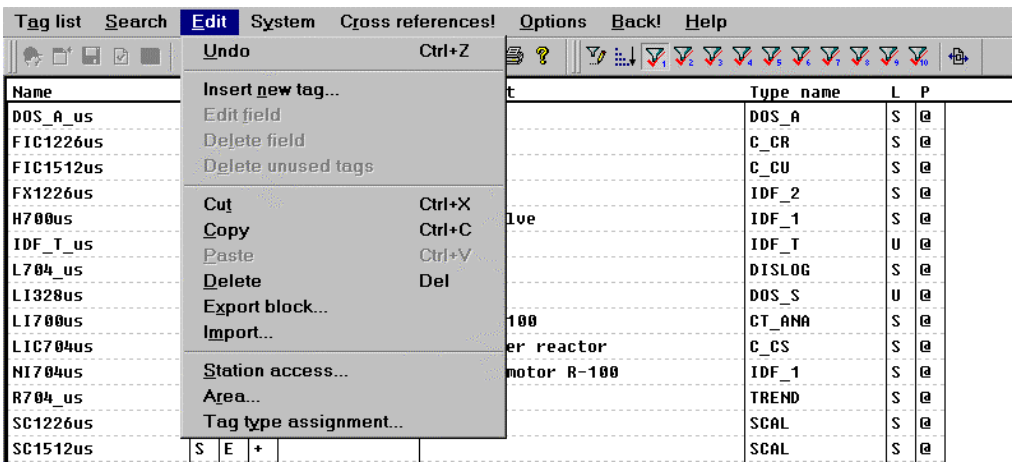
Tags for which access rights have been assigned via a gateway, but which are not being used in a program, count as **unused tags**.

Under Project Options on the main menu you can set a parameter to specify whether or not the **search filter activation** is retained on exiting the variables list. The **search filters configuration** is stored along with the project.
Activated search criteria can be recognized in the status line by the number of entries displayed and by the fact that the corresponding toolbar button is depressed. The configured search criteria are displayed at the toolbar buttons as tool tips.

C 4.5 Edit tag list

→ *Edit*

Various menu options are available for editing the individual tag list items. You can undo an action, insert new tags, delete, cut or copy items, etc. Also, text blocks can be imported or exported.



di0322uk.bmp

C 4.5.1 Undo

→ *Edit* → *Undo*

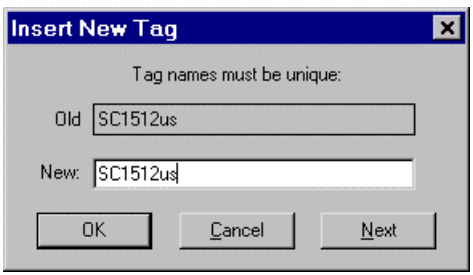
The last change is withdrawn and the text is shown as it was before the last change. If it is not possible to undo something, then the menu item cannot be selected (reverse highlight).

C 4.5.2 Insert new tag

→ *Edit* → *Insert new tag*

If the cursor is located on an empty field, e.g. at the end of the list, a new tag may be entered directly into the individual fields in this line of the list.

If the cursor is on a list entry, a window will appear. The selected name appears as the default for the old name and the new name. The new name must then be **changed** by entering the desired new name. All the other data is taken over from the tag which was selected previously.



di0336uk.bmp

Old	The name of the selected tag for information only.
New	This shows the name of the selected tag as the default and may be changed by entering the desired new name.
OK	The tag selected is taken over.
CANCEL	The existing tag is not changed.
NEXT	Not assigned for tag entries. Jumps during import to the next line in the list that is already occupied.

When searching is activated, i.e. the list is not displayed fully, it is not possible to enter any new tags. When inserting a block, the tags inserted must be assigned new names. The NEXT button enables certain tags within the block to be skipped and not be re-included in the list.

C 4.5.3 Edit field

→Select desired field with double click (highlight box)
The cursor appears at the last item of entry
→Cursor click on item of entry in field
→Enter changes

The text contents of the selected field may be changed. After the change has been made, a further window may appear, requesting confirmation of whether the change shall apply throughout the project or just in specific programs. **See page C-8, Changing tag list settings**.

C 4.6 Delete field

Entries in the fields **Name**, **T, A**, **R**, **Type name**, **L** and **P** cannot be deleted with this command.

The tag may be deleted by selecting an entire line in a list.

→Select desired field (highlight box, cursor appears at the last item of entry)
→DELETE button

C 4.7 Delete unused tags

Unused measuring points become **red** color in the tag list

The unused tags can be shown or hidden in the search filter for the tag list.

Even tags for which access rights have been assigned via a gateway, but which are not being used in a program, count as **unused tags**.

C 4.8 Edit block

Block must be defined individually. A block consists of a row of full lines in a list which has been marked. It may be marked as follows:

→Click cursor to move to where the block is to start → Hold down the left mouse button and drag the mouse to the end of the block to mark it.

When the entire block is marked, release the left mouse button or SHIFT key. The blocks may now be labeled and saved for future use.

C 4.8.1 Cut

Select block → *Edit* → *Cut*

Text blocks which have been defined are removed from the text and saved in the buffer.
The text in the buffer can then be incorporated again at any item using the *Insert* command.

C 4.8.2 Copy

Select block → *Edit* → *Copy*

Text blocks which have been defined are copied and saved in the buffer.
This text may then be incorporated again at any item using the *Insert* command.

C 4.8.3 Paste

Select block → *Edit* → *Paste*

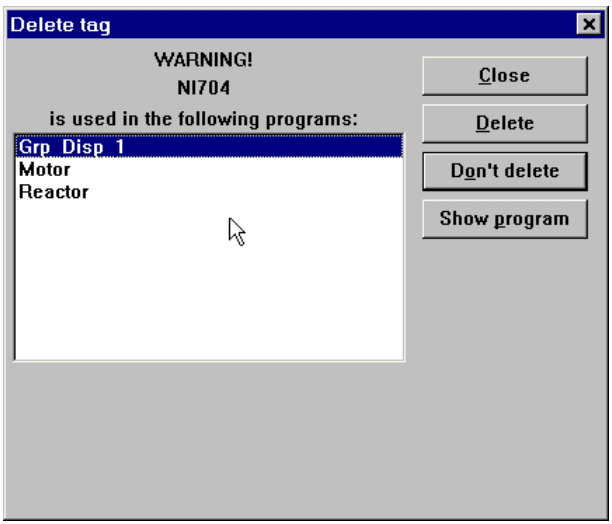
A text block which has been copied or cut is saved in the buffer and may be inserted at the desired item marked by the cursor.

Variable names or tag names have to be changed, so the same window appears as in the menu item *Insert new tag*.

C 4.8.4 Delete

Select block → *Edit* → *Delete*

A text block which has been defined will be deleted from the text after a query.



di0355uk.bmp

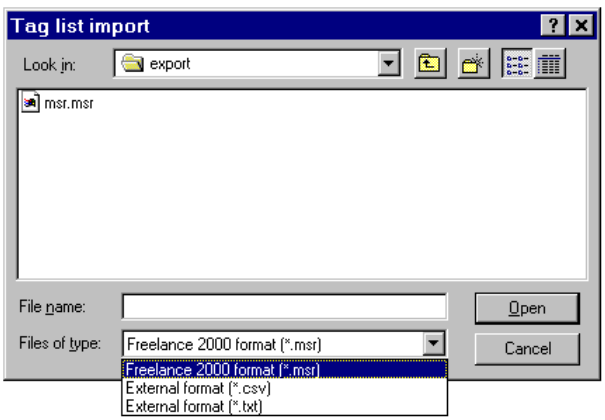
DON´T DELETE	Selected variable/tag is not deleted
DELETE	Selected variable/tag is deleted
SHOW PROGRAM	Go to selected program

C 4.8.5 Import

→ *Edit* → *Import*

A file stored using File export can be copied back from a data carrier (hard disk or floppy). Another window is displayed in which the path, filename and file type should be entered.

If, when importing tags to a project, tag names are found which are the same as tag names already existing in the project, those being imported will be treated as new tags. See page **C-14**, **Insert new tag**.
Files exported using DigiTool or text files can be imported depending on the file types concerned.



tk001us.bmp

Freelance 2000 format
Files exported from the tag list using *File export* can be re-imported. The files have the extension **msr**.

External format
Tag import also offers the option of importing files into the tag list that have been created by external applications (e.g. Microsoft Excel). These files have the extension **csv** or **txt**.

The files for import must be text files in Unicode format, and they should be structured as follows: Each line of the import file should contain the tag name, short text and long text. The tag type is not yet defined at this point.

These three text items are separated by a list separator, which can be either a comma ‘,’ or a semicolon ‘;’. The different separators must not be combined within the same line. In the event that a text item itself includes the list separator, the text or the list separator should be enclosed in quotation marks (“ “).

Example:

TIC1304;Boiler 4;Temp. boiler 4
FIC1205;Air ";" K5;"Air flow; K5"
FIC1204;Air K4;Air flow K4

☞ The import file must be a Unicode file.

When files are imported the system checks to see whether tag names conform to the existing criteria; any tag that does not conform will not be accepted for import.

See also **Engineering Manual, System Configuration**, **Project Manager**, **Project management Options**, **Expanding tag names**.

The short text must be no more than 12 characters in length. If it is any longer than this, characters after the twelfth one will be ignored during import. The long text can be a maximum of 30 characters in length, and, like the short text, no more than the maximum permissible number of characters will be read in during import.
Neither the short text nor the long text may contain list separator characters unless these are enclosed in quotation marks. Failure to observe this rule will result in the tag import routine interpreting the character as a separator and the following character as the start of the following field. If a line contains more than two list separators, then any characters between the third list separator and the end of the line will be ignored. If, on the other hand, a line contains less than two separators, then the tag will be rejected and not imported.

☞ If format errors are detected in the import file during import, then the import process will be abandoned at that point.

When new tags are being entered, tag type defaults to '----' and library type to '-'. The tag type can be assigned directly in the tag list (**see page C-8, Changing tag list settings**) or alternatively this can be done while configuring the blocks.

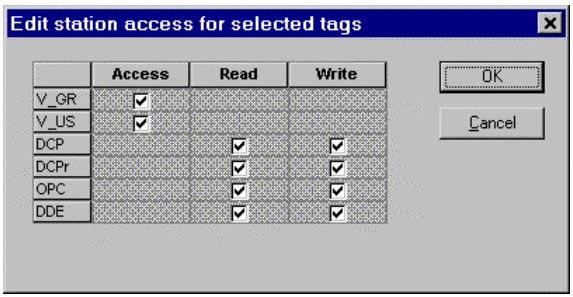
C 4.8.6 Export block

Select block → *Edit* → *Export block*

A block which has been defined may be saved as a file on data media (hard disk, floppy disk). Another window appears for the file path and filename specification. This file may then be read from other projects outside the parameters of this project via the *Import data* command.

C 4.8.7 Station access

→ Select block → *Edit* → *Station access*



di0321uk.bmp

If the inputs, outputs and parameters of a variable are to be read or written via a gateway, this access must be released

- in the project tree on the resource and
- in the tag list.

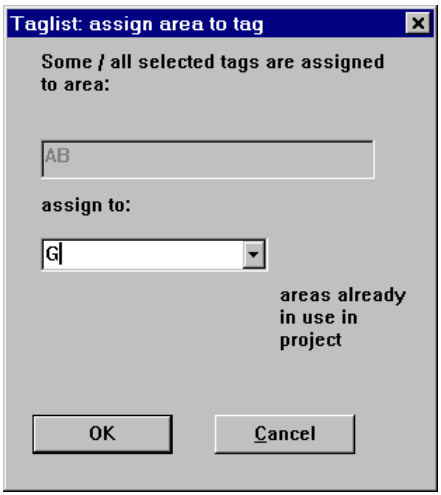
Also refer to the manual **DigiDDE32**, and the manual **Coupling Freelance 2000 - Maestro UX**.

For each operator station, certain tags which should not be operated on this station can also be filtered. If no access is released, this tag can no longer be selected from the tag list.

See also **page C-10, Normal view and station view**.

C 4.8.8 Plant areas

→ Mark block → *Edit* → *Plant areas*



tk002us.bmp

All the plant areas already assigned in the marked block are shown.
All the tags in the marked block are assigned to the plant area entered here.

C 4.8.9 Assign block

→Mark block → *Edit* → *Tag type assignment*

A new block type can be allocated to the marked tags. All the block types recognized within the system are available for selection: block types from the standard library, user-defined blocks and types from the special library. **See page C-8, Changing tag list settings.**

C 4.8.10 Access rights

 Select block → *Edit* → *Access rights*

If the add-on package DigiLock is installed, individual tags or selected blocks of tags can be locked here for certain user groups. On the relevant operator station the tag can then only be inspected, or also controlled, or not called at all.

See manual **DigiLock.**

C 4.8.11 User groups

 Select block → *Edit* → *User groups*

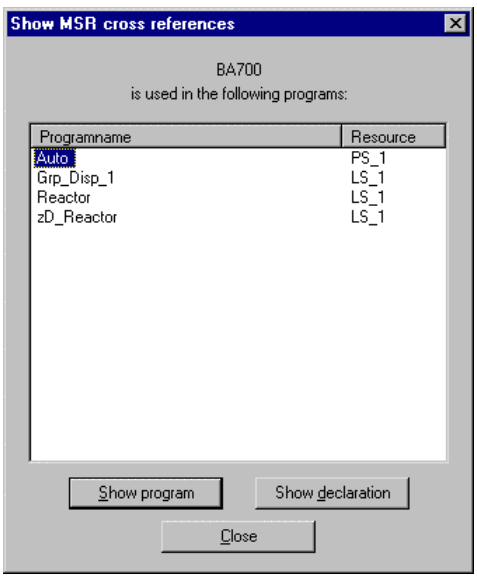
If the add-on package DigiLock is installed, individual user groups can be assigned here to certain resources. See manual **DigiLock**

C 4.9 Cross references

Cross references of a tag can be displayed in a list by means of *Cross references.* Cross references are references relating to these tags in programs, displays, listings etc., i.e. locations where these tags are used.

Select field → *Cross References*

A window displays the names of the relevant programs



di0356uk.bmp

SHOW PROGRAM Calling a program with pre-selection of this tag or calling the module to which the tag is allocated.

SHOW DECLARATION No function

C 4.10 Hardcopy

 → *Options* → *Hardcopy*

The screen contents may be output to a printer. A window shows the printer in use, print quality and the number of copies required. Changes to these settings may be made by selecting SETUP.

C 4.11 Set colors

 → *Options* → *Colors*

It is also possible in the tag list to freely specify color preferences for any unused tags.

C 4.12 Save column settings

 → *Options* → *Save column settings*

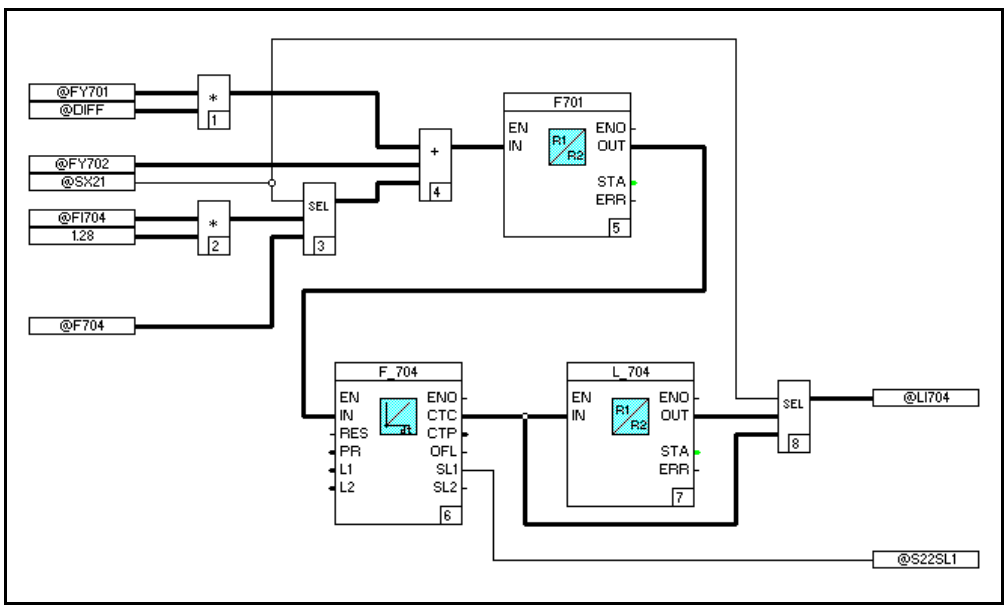
The column width setting is stored.

C 4.13 Back

 → *Back!*

Back to previous screen.

D Function Block Diagram (FBD)



Contents

D 1 General Description - Function Block Diagram

Function block diagram (FBD) is a graphically oriented IEC 61131-3 programming language. The graphic representation of FBD programs has been completely reworked and brought into alignment with the Ladder Diagram (LD) editor.

FBD's CAD functionality permits simple positioning and connecting of functions, function blocks and their variables.

The working area of an FBD is laid out on 10 x 10 screen pages. The individual pages can be accessed via vertical and horizontal scrolling. The entire work area is covered by a grid. The divisions between the indivudual pages are shown as dotted lines on the screen. The printed form of the program contains page-for-page exactly what is seen on the screen.

An FBD program consists of the following **graphic elements**:
- Connections and lines
- Variables and constants
- Functions and function blocks

The signal flow of a FBD is from left to right. The signal flow lines are edited either by pressing the CTRL key or alternatively by activating an appropriate "Line drawing" mode.

The named variables can be either selected from the list of system-wide variables and copied in, or declared directly in the program. See **chapter Variables**.

In FBD programs the processing sequence of the blocks can be set individually.

As extension of the IEC language definition, any structured data type can be used. Structured data types can be used in data exchange with other stations on the DigiNet S bus or with TCP/IP send and receive blocks.

If communication with the process stations has already been established when FDB programs are loaded, the editor goes into a special mode which permits display of the current values.
See also **Engineering Manual, System Configuration, Commissioning**.

D 1.1 Creating an FBD program

An FBD program is created in the project tree**.**

Project tree → Select insert position in the project tree
→ *Edit* → *Insert above*, *Insert below* or *Insert next level*
→ FBD program from "Object selection"
→ Assign program name and possibly a short comment

Each new FBD program has a blank graphic region and no comment, the processing state **incorrect** and the generation date as its version code.
The name of the program list (PL) is preset as program name. The short comment of the program is taken over and can be changed.

D 1.2 Calling the program

A program can be called up with:

Project tree → *Edit* → *Program*
or double click program

The program is displayed with its contents (functions, signal flow lines, etc.), and can be changed.

D 2 Structure of the Function Block Diagram

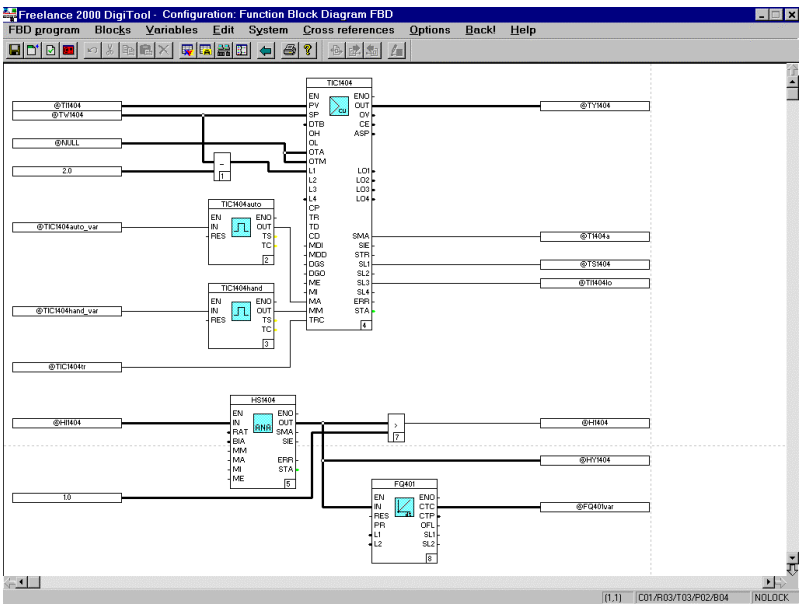
D 2.1 Function block diagram user interface

The operator interface of an FBD program consists of:

Menu line

Graphic region

State line



tl100us.bmp

Graphic region — The function blocks and signal flow lines are programmed in the graphic area of the FBD program.
The graphic region provides a grid in order to facilitate positioning of the blocks, while observing minimum distances. The user can place the corners of blocks and signal flow lines only on the grid lines. Variables and constants can be placed anywhere in the program; they are displayed and edited in rectangles. The grid can be switched on and off.
A FBD can be up to 10 x10 pages long. The individual pages are delimited with dotted horizontal lines. Care should be taken not to position objects on the dotted lines , as such objects will be split onto separate pages when printed in the documentation.

State line — Displays the current program state.

D 2.2 FBD program menu structure

FBD Program	*Save*
	Documentation
	Check
	Header
	Comment
	Exit
Blocks	*Analog*
	Binary
	Constant
	Converter
	Acquisition
	Arithmetic
	Controller
	Standard
	Open loop control
	Monitoring
	Modbus Master
	Modbus Slave
	System functions
	TCP/IP Send and Receive
	DigiBatch
	User function blocks
	User menu
Variables	*read*
	write

Edit	*Undo*
	Change data
	Parameters
	Processing sequence
	Change number of inputs
	Zoom to user FB
	Select variable
	Toggle use of process image
	Cut
	Copy
	Paste
	Delete
	Export block
	Import block
	Draw line
System	*Variable list*
	Tag list
	Hardware structure
	Structured data types
Cross references	*Cross references*
	Find next
	Find previous
Options	*Version*
	Hardcopy
	Raster on
	Define user menu
	Colors
Back!	
Help	*Contents*
	Overview
	Using help
	About

D 2.3 Modifying default settings

D 2.3.1 Switching on and off the grid

Options → *Raster on/off* (→ *Save!*)

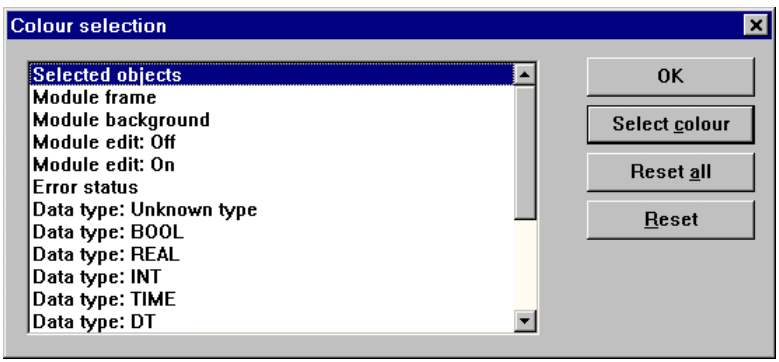
The positioning grid in the graphic region is switched on, if it had been switched off, or vice versa.

The changes made to the settings are preserved until another window is opened. If the settings are to be preserved for a longer period, **save** the program after making modifications.

The saved settings of the last program processed are preset. The grid is switched on for the first program to be generated for a new project. The grid spacing cannot be changed.

D 2.3.2 Changing the program colors

Options → *Colors*
→ Select object for which the color is to be changed
(e.g. color of a block background)
→ Select color



di0136uk.bmp

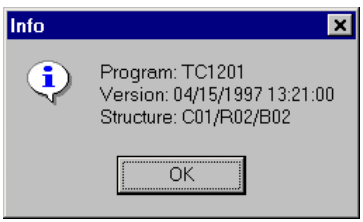
SELECT COLOR	The color for the selected object can be chosen. The default color is marked.
RESET	The color of the selected object returns to the default value.
RESET ALL	The colors of all objects are reset to the default values.

D 2.4 Displaying program information

D 2.4.1 Program version and assignment to the project

Options → *Version*

The program name, date of last program modification (version) and program assignment to the project, resource, task and program list are displayed.
The program assignment can be displayed as **long** or **short text**, with the necessary adjustment being made in the project tree under options.



di0130uk.bmp

D 2.4.2 Program state

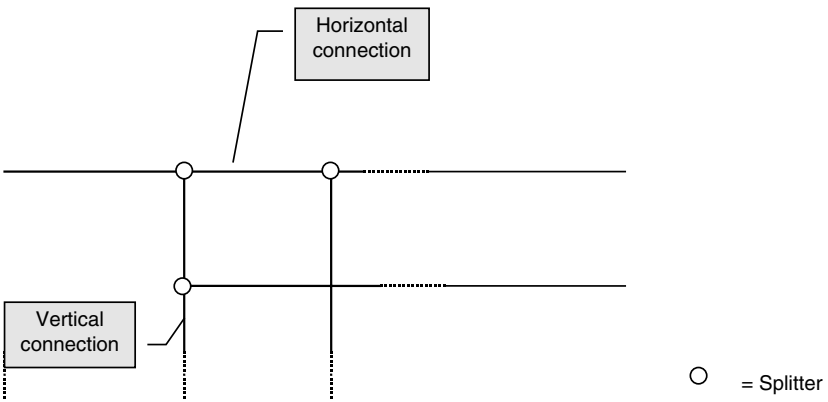
The **state line** indicates the name of the program currently being edited, the editor position and the current user.

Editor Position (4,1) Shows page (line, column) currently being edited.

D 3 Description of FBD Program Elements

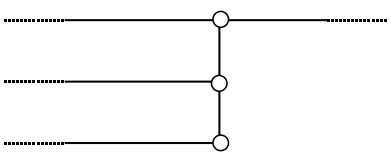
D 3.1 Connections and Lines

Horizontal and then vertical connections can be made to variables and blocks. Connections are shown as horizontal or vertical lines.



Function	Description
horizontal connection	Transports the condition from the left end to the right end.
vertical connection	Distributes the conditions from the horizontal connections on the left to other horizontal connections on the right.

In an FBD program, it is not possible to join multiple horizontal connections together to form a single horizontal connection.

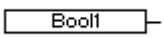
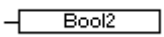
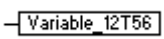
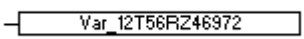
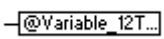
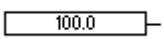


D 3.2 Variables and Constants

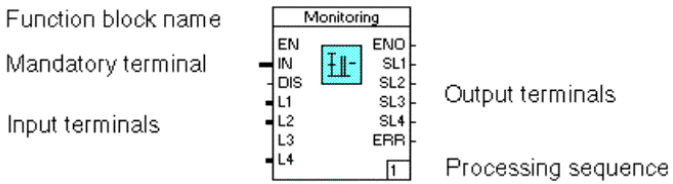
Variables and constants can be placed anywhere in the program, and are displayed and/or edited in a rectangle.

Containers for variables and constants have a short and a long version. The short version can display 10 characters. If the label is longer than 10 characters, the overflow is indicated by '….' The long version can accommodate the maximum possible length of label.

Variables can be read and written either via the process image or directly. Reading or writing via the process image is indicated by @.
Since variables can be placed anywhere in the program, it is essential when inserting them to specify whether they are to be used for reading or writing. Depending on whether a variable or constant is to be used for reading or writing, the surrounding rectangle is provided with either an input or output pin of the appropriate data type.

Symbol	Description/function
	**Variable for reading**
	**Variable for writing**
	**Short version** At most 10 characters can be displayed Overflow indication '…'
	**Long version** Max. possible label length
	Read/write via **process image**
	**REAL Constant**

D 3.3 Blocks



di0150us.bmp

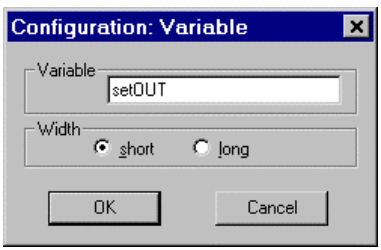
Frame	The block frame limits the selector area of the block. From its color one can establish whether the block has been selected or parameterized in any incorrect manner. The color display can be changed here. See **Page D-9, Changing the program colors.**
Function block name	Unlike the functions, all function blocks are displayed with a **tag name** (max. 16 characters). All block names can be found in the systemwide **tag list**. The color in which the block name is written denotes the processing state (enable / disable) and can be set likewise.
Icons	An icon is used to symbolize the block type of a function block - a function abbreviation that of a function.
Terminals	Here a distinction must be made between inputs and outputs. Corresponding to the signal flow, inputs are always displayed on the left and outputs on the right. Just as in the case of the signal flow lines, the **color and line width** of the terminals reveal information on the data type needed/set.
Mandatory/ Optional parameters	Mandatory terminals call for data supply via the signal flow line in order to enable the block to operate correctly, while this does not apply for optional connections. Optional terminals are displayed shorter for the purpose of differentiation. Some optional terminals disappear altogether due to the parameter definition of fixed values.
Terminal designation	An abbreviation next to each function block terminal denotes the terminal's function, for example **EN** for enable**.**
Processing sequence	The code on the lower right of the block indicates the processing sequence within the program.

D 4 Parameterization of FBD program variables

FBD elements are parameterized by selecting the element and then carrying out one of the following actions.

Edit → Parameters
→ Double click the element
→ Right click in the graphic region and choose parameters on the context menu.

D 4.1 Parameterize Variables



tl024us.bmp

Variable Name of variable
A variable can be selected from the variable list with F2.

Width

short The short version, in which only 10 characters can be displayed, will be chosen for the variable. If the label is longer than 10 characters, the overflow is indicated by '….'

long The long version, which can accommodate maximum-length labels, will be chosen for the variable.

D 4.2 Parameter definition of function blocks

D 4.2.1 Parameter types

The specifications needed for editing and displaying a block in the Freelance 2000 system are called parameters, with a distinction being made between the following types:

Mandatory parameters are essential parameters such as the block name and depending on the block type, the parameters of certain inputs and outputs.

Optional parameters are not absolutely necessary parameters e.g. short text, long text, dimension, access facility, message value. They always feature default values on first positioning the function block.

External parameters — are assigned to a block and vice versa on connecting a signal flow line.

Internal parameters — must be entered within a parameter definition mask. They include information such as the block name and limit values.

D 4.2.2 Calling parameter definition masks


Select the function block to be parameterized
→ *Edit* → *Parameters*


Double click function block

Changeover is effected to the first parameter definition mask of a function block. All other selected elements are automatically deselected. After return from the parameter definition mask, the function block of the modified parameter definition mask is accordingly depicted again.

D 4.2.3 Enter mandatory parameters

The **mandatory parameters** of the individual function blocks of this program must be entered in order to be able to terminate an FBD program correctly. All mandatory parameters feature a **red** background in the parameter definition mask. Generally, this applies only to the **block name** (max. 16 characters) of a function block.

All block names entered for function blocks are summarized systemwide in the tag list. For a description see **chapter Tags**.

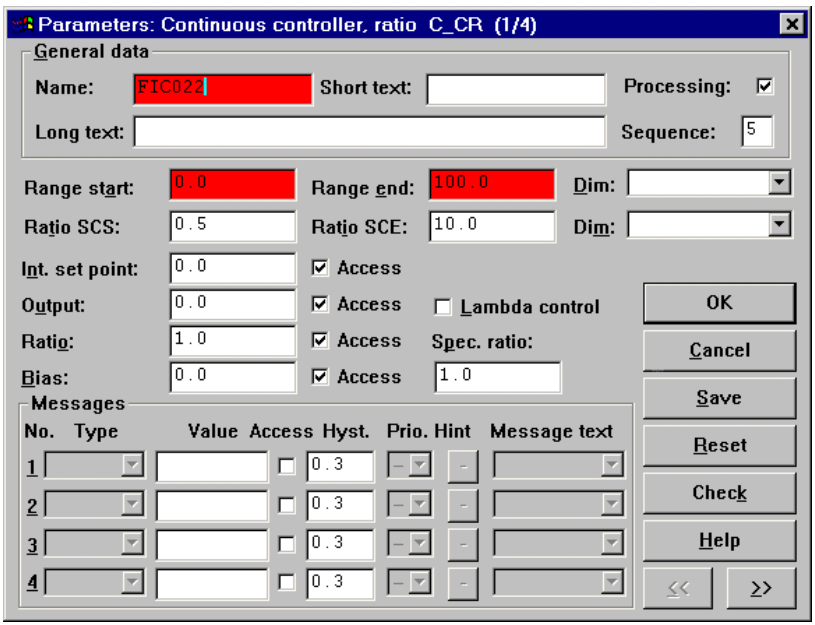
Alternative input possibility for the block name


Select text field *Name:* → F2
→ Select block name from tag list

D 4.2.4 Handling the parameter definition masks

By virtue of the different parameters governing the various function blocks, there is no uniform parameter definition mask. However, certain sections are used similarly in all or in some parameter definition masks. Besides, there are several parameter definition masks for large blocks and they can be edited in any order desired.

Using the parameter definition masks of the function block "Continuous ratio controller C_CR" the basic features are outlined below:



di0627uk.bmp

General data — Name, short designation of block, if necessary number of parameter definition mask currently in use.

Group — Some parameters are classified in groups e.g. the message values. The parameters are placed in a frame and a group name portrays the parameter function in the upper frame corner.

Input field color — Red background: Mandatory parameters
Blue background: Marked for overwrite

Text field — For entering block name and long text for example. If the cursor is moved to a text field with the tabulator key, the field is marked for overwrite (mark by double clicking when using a mouse).

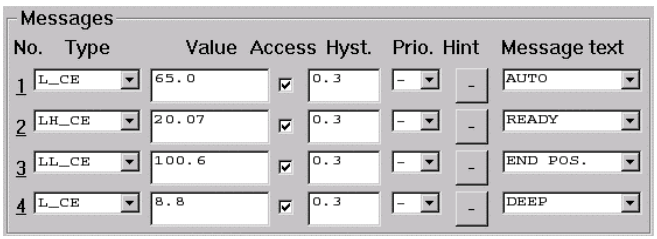
☞ The block name can also be selected from the tag list via the function key F2.

The optional parameters, short and long text, can only be entered after assigning a block name.

Data field — For example for entering parameters such as measuring range start and measuring range end. In the case of parameters that can also be specified externally, data can only be entered if no signal flow line is connected to the respective terminal. Conversely, the terminal disappears from the

block display if a parameter has been entered. Please consult the block description for the parameters to which this applies.

List



di0165us.bmp

There are lists where only the preset list entry is visible. The invisible list section can be opened out (arrow). The desired entry is taken over by clicking the input field.

Some lists have an input field that can be freely edited. In these cases, the arrow points away a little from the input field. The entered text is taken over into the list and is available also in all subsequent parameter definition masks of blocks of the same type.

Virtual keys see **chapter Getting Started: DigiTool.**

To enhance transparency, short and long texts should be entered for the blocks.
The parameters featured in the block description are preset. See **Engineering Reference Manual, Functions and Function Blocks**.

A block input or output that is linked with a signal flow line cannot be assigned internal parameters and vice versa.
Short and long text can only be entered after allotting one of the block names.

D 4.3 Change the processing sequence of the blocks

Select block → *Edit* → *Processing sequence* → Enter processing number in block (the old one is marked for overwrite).

If blocks feature a parameter definition mask, the processing order can be changed there, too.

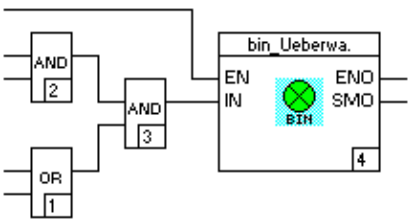
Press CTRL and hold
→ Click the processing number on the lower right of the block with the mouse.

The unambiguous order in which the program blocks are processed during program execution is changed.

The processed block is given the newly entered processing number. The processing numbers of all other program blocks are corrected so that their mutual order is preserved and no blanks appear in the order. If a number that exceeds the total number of blocks used in the program is entered, the edited block is given the total number as its processing number.

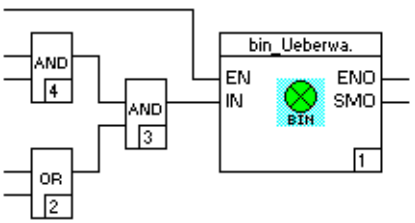
The processing number is assigned automatically in the chronological order in which the blocks are positioned.

☞ Since the blocks are generally not placed in the program in the order in which they are to be processed during operation, it is advisable that all blocks be checked after linking and the order changed if necessary.



di0143.bmp

Reasonable order



di0142.bmp

Unreasonable order

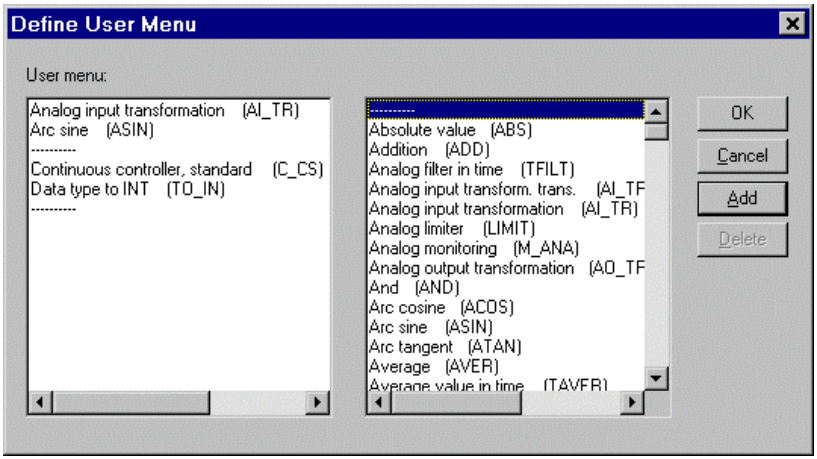
D 4.4 Define user menu

Functions and function blocks that are required frequently for creating projects can be grouped together in a separate block menu for the purpose of clarity.

Add blocks

Options → *Define user menu*
→ Select function block in the right-hand window list
→ ADD or double click

☞ The selected block or the separation lines for **combining blocks to form groups** are inserted accordingly at the point at which the separate block menu currently ends.



di0128uk.bmp

The left-hand list in the window displays all blocks entered into the user menu. The right-hand list contains all blocks available for selection in the libraries.
Horizontal lines can be inserted into the block's user menu to visually separate blocks.

Resorting blocks

Options → *Define user menu* → Click on block in left window → Move mouse in between two block entries → Single click

Deleting blocks

Options → *Define user menu* → Select block in the left-hand list → Delete

The block is deleted from the user menu.

D 5 Editing FBD Programs

D 5.1 Drawing signal flow lines

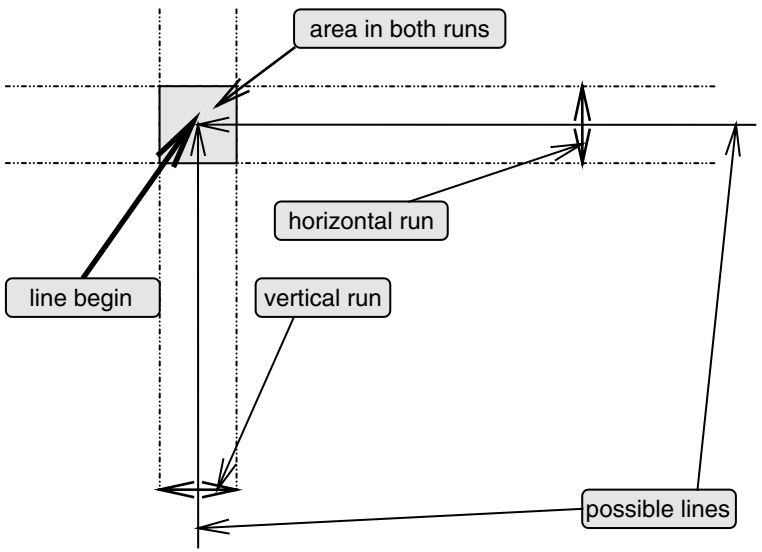
The FBD editor has a special line-draw mode to enable the drawing of horizontal and vertical signal flow lines. Line-draw mode is activated as follows:

→ *Edit* → *Draw lines*
or
Right click (context menu) → *Draw lines*
(the mouse pointer will change to a cross)

A single mouse-click determines the beginning of the line. A horizontal or vertical line is drawn by moving the mouse, as long as the cursor does not leave the line run or cross a block boundary.

After a second mouse-click, both the end and the beginning of the new line are determined. Clicking on a point lying in both the horizontal and vertical run, or clicking outside either run, aborts the line.

The illustration below shows the line draw mode. The run is exactly two grid units in width.



Draw line

Mouse-clicks for beginning and end of line
or
press CTRL and left mouse button together

Deactivate line-draw mode:


Right-click or Esc key

Display signal flow lines

The signal flow lines indicate the data type transported. If the signal flow line has the processing state **selected**, **incorrect** or **not linked**, this is indicated.

The state or transported data type of the signal flow line can be recognized from the line width and color, while the color can be set as desired by the user (see **page D-9, Changing the program color**).

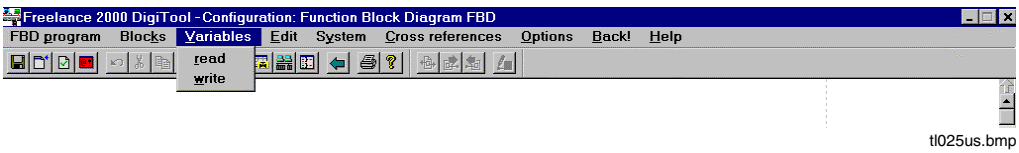
The relation between data type, processing state, line width and preset color are shown in the following diagram:

BOOL	black	narrow	
BYTE	gray	wide	
DINT	grass-green	wide	
DT	dark yellow	wide	
DWORD	magenta	wide	
INT	light green	wide	
REAL	black	wide	
TIME	light yellow	wide	
UDINT	brown	wide	
UINT	turquoise	wide	
WORD	dark blue	wide	
STRING	black	wide	
STRUCT	black	wide	
Error state	red	narrow	
selected objects	turquoise		
not connected	black	narrow	

di0152gr.bmp

D 5.2 Inserting variables and blocks

Variables → Choice of elements to be inserted
Blocks → Choice of elements to be inserted



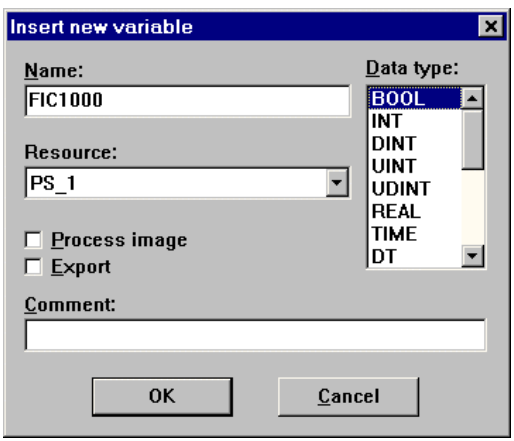
tl025us.bmp

After the element to be inserted is chosen, the cursor takes on the shape of the selected element. The selected element can be positioned as desired (with a mouse-click). If it does not fit, the cursor remains an outline and a warning tone is sounded.

If the placement was successful, the outline cursor remains, and additional elements of the same kind as the one just selected may be inserted.

The insertion operation is ended with a right mouse-click.

D 5.2.1 Inserting variables



di0133uk.bmp

Resource Sets the allocation of variables to resources. Each variable must be allocated to exactly one resource. Other variables have read-only access to these variables.

Process image ☑ The variable is to be accessed by the process image. The process image is an integral part of the task and is updated at the beginning and end of the task execution cycle. See also **Engineering Manual, System Configuration**, **Project Tree, Process Image**.

Export ☑ The variable is to be read by other resources.

If a variable is used for the first time in the project, it is taken over automatically into the system-wide variable list. See **chapter Variables**.

Multiple reading use of the same variable in a program results in a warning but is permissible.
Multiple writing use of the same variable in a program is not permissible and results in an error.

Previously defined variables and I/O components can be selected from the list directly.

→ F2 key
→ Select one of the variables or I/O components already existing in the project from the list

If the variable already exists in the project, the entries in the window "Insert new variable" are omitted.

An I/O component can only be exported via a variable, never directly; i.e. , the I/O component can not be read in other resources by means of the component name. Note that variables which are to be allocated to an I/O component do not feature gateway write rights.

D 5.2.2 Selecting and positioning blocks in the program

Blocks → choose the desired block type
→ move to the desired location in the graphic region with the mouse
→ place with the left mouse button (for blocks with a variable number of inputs, the size must now be set by using the mouse to make a vertical adjustment; confirm with a left click).
→ either position another block of the same type, or
→ end positioning at any time with ESC or with the right mouse button.

After a block is selected, it can be inserted in the graphic region. While it is being positioned, it is displayed schematically. After it is positioned, a new outline is used to indicate that another block of the same type can be inserted.
Blocks with a selectable number of inputs (for example, AND, OR or EXOR) are displayed in minimal size during positioning. After they are placed, their size can be changed at once: when pulled vertically with the mouse more inputs become visible.

The new block will have the lowest processing sequence number not yet assigned in this program. Blocks that take parameters have a parameter mask with default values but no block name.

The screen representation of the block must not cover other program elements. A minimum distance of three grid units for input or output pins and two grid units for other blocks must be maintained.

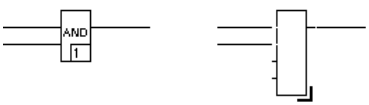
D 5.3 Change number of inputs

Select block → *Edit* → *Change number of inputs*
→ The mouse must be moved up or down until the required number of inputs is displayed → Confirm.
→ End positioning at any time with Esc or the right mouse button.

Double click the lower boundary line of the block
→ The mouse must be moved upwards or downwards until the required number of inputs is displayed → Confirm.

The number of input terminals of the function block will be changed.

The function block terminals already connected are firmly positioned and are not moved by changing the number of inputs. Hence the number of inputs can be changed without affecting the terminals already connected.



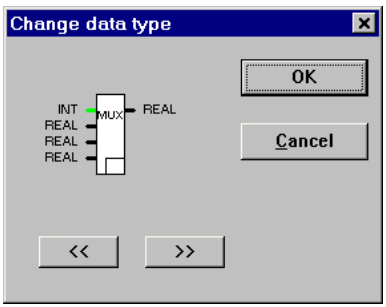
If the procedure is interrupted, the block retains its old state.

Changing the number of inputs of the selected block must be permissible, as with AND, OR and EXOR, for example. Assignment of inputs to the blocks (analog, binary, etc.) can be seen in **the Engineering Reference Manual, Functions and Function Blocks.**

If inputs already connected but no longer needed are to be removed from a block, the signal flow lines belonging to the inputs must first be disconnected from the block.

D 5.4 Display and change data types

Select block → *Edit* → *Change data type*
→ Set and enter the required data type with >> and << .



di0131uk.bmp

The data type of the block terminals are displayed in text and graphically. On changing, the display is adapted to the new data types. The display of connected signal flow lines changes accordingly.

Please consult the respective block descriptions for the data types possible for each block. **See the Engineering Reference Manual, Functions and Function Blocks**.

The data types of the selected block can only be changed if the block permits other data types. They can only be changed identically for all terminals. Irrespective of this, some data types can also be converted using the converter blocks **_to_** and *Trunc*.

D 5.5 Inverting a block terminal

Keep CTRL pressed and click the block terminal to be inverted with the left mouse button.

A negation is set or reset, for the selected terminal.
Added inversion markers are treated as a component of the function block.

All blocks have non-negated terminals as default.

Block with negated terminal

The block connection to be inverted must be of the BOOL (binary) data type.

D 5.6 Changing variables

Double click the variable to be changed
→ By clicking again, the variable can be marked for overwrite
→ Change variable name → ENTER
→ Select the data type in the window "Insert new variable"

The window entries are omitted if the variable already existed in the project.

Double click the variable to be changed → F2 → Select one of the variables already existing in the project in the window "Select Variable/Component".

Position the cursor with TABULATOR KEY and ARROW KEYS on the variable to be changed
→ ENTER
→ with Shift + End the variable can be marked for overwrite
→ Change variable name → ENTER
→ Window "Insert new variable"

The new variable name is taken over into the program and variable list. The old variable remains in the variable list.

If the modified variable had been used in several programs of the project, they are not affected.

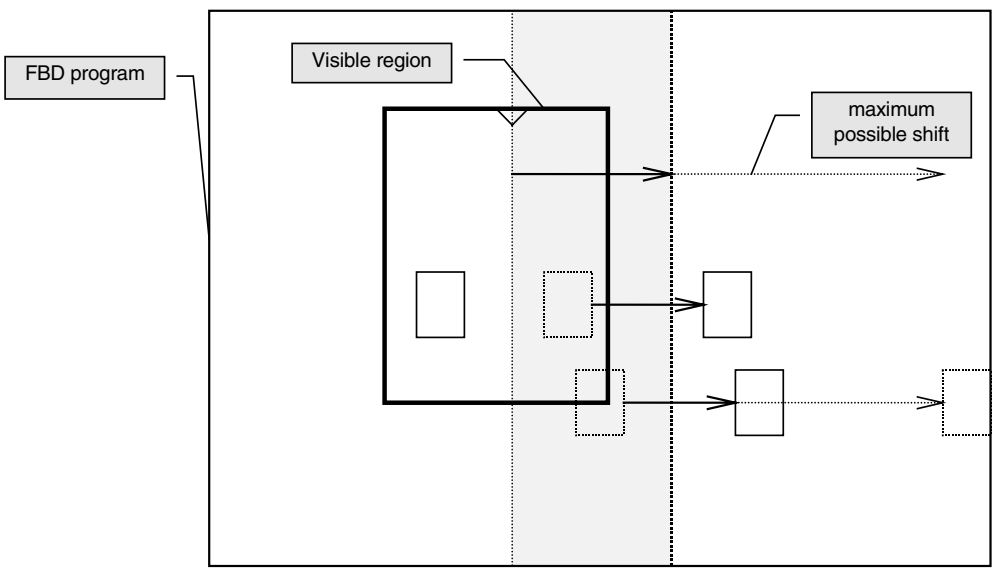
D 5.7 Inserting columns and rows

Inserting columns

Insertion of columns relates only to the current program. Clicking the left mouse button on the upper or lower border of the display will bring up a triangle pointing up or down respectively and with a horizontal dashed line. This triangle can be shifted to the right in grid-unit steps.

Shifting the triangle has the effect of inserting a corresponding number of columns and shifting the partial network to the right of the vertical line by a corresponding number of grid units to the right.

If the mouse moves as far as the edge of the visible part of the display, then the visible region scrolls. The triangle can only be moved if the partial network to be moved is not touching the right-hand edge of the program and if the vertical line does not intersect a network element other than a horizontal link. The diagram below should further clarify the procedure for inserting columns.



Horizontal links are extended accordingly when columns are inserted.

Inserting rows

Rows are inserted in the same way as columns. The triangle appears at the left- or right-hand edge of the visible area. The movement markers run in a horizontal direction. When rows are inserted, vertical lines are extended accordingly.

D 5.8 Block operations

D 5.8.1 Select program elements

Select individual program elements

Select by Left click on the required program element.

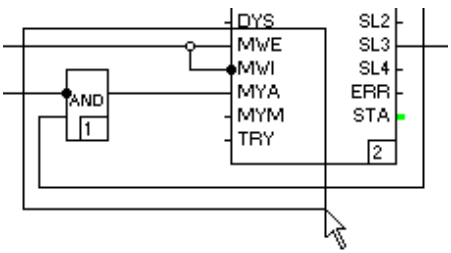
The entire surface of the program element is valid as selector field. The program element is selected for further processing and shown accordingly.

The non-selected state is preset.
Inversions and link points of signal flow lines are never displayed as selected.

Select several program elements concurrently

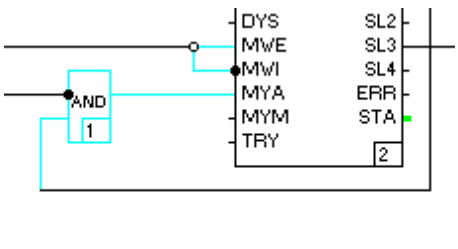
Place a frame around the elements to be selected with pressed left mouse button

All elements fully enclosed in the frame are **selected concurrently** and shown accordingly. In the case of the signal flow lines, this applies for all segments fully enclosed in the frame. After selection, the desired operation can now be performed as in the case of single elements. For example: *Edit* → *Cut*.



di0147.bmp

Place a frame



di0148.bmp

Selected program elements

Select additional program elements

Press SHIFT and hold → Select another element

One element is selected in addition to the existing selection and is shown accordingly.

It is also possible to select several elements via SHIFT and "place in a frame".

D 5.8.2 Deselecting program elements

Deselect all selected program elements


Left click on a free point in the graphic region or selection of a non-selected element.


With the ARROW KEYS, move the cursor to a free point in the program → press SPACE.

The program elements are deselected and shown accordingly.
A selection is canceled automatically on opening another window.

Deselect individual program elements of a selection


Press SHIFT and hold and click element to be selected.


With the ARROW KEYS, move the cursor to the element to be deselected
→ SHIFT + SPACE.

An element of the already existing selection is deselected and shown accordingly.

D 5.8.3 Copy


Context menu (right mouse button) → *Copy*
or
Edit → *Copy*


Press key combination *Ctrl-C*

Copy has the effect of transferring the selected elements to an internal storage location. Elements transferred there through a previous *Copy* are overwritten. Whether or not there are currently any elements in the internal storage can be seen from the menu choice *Insert* in the *Edit* or Context menu. If this menu choice is disabled, this indicates that the internal storage is empty.


The selected elements can only be copied within the same FBD program. It is not possible to copy them into a different FBD program, as that would involve quitting the FBD editor in order to call up another FBD program via the project tree.

If a block is required for use in another FBD program, it must be exported and then re-imported into the target program.
When function blocks are copied, the parameter data remain unchanged. However, the tag name is deleted in the copy, as it must be unique.

D 5.8.4 Cut / Delete


Context menu → *Cut* or *Delete*
or
Edit → *Cut* or *Delete*


Press SHIFT-DEL to cut, or DEL to delete

If the selected elements have been cut, they can then be re-inserted in the program using *Paste*. However, just as with *copy,* this applies only to within the same program. *Cut* has the effect of overwriting any elements held in the internal storage at the time.


If elements are deleted, they cannot subsequently be pasted. Deleted elements can only be restored by quitting the program without saving.

When function blocks are cut, their parameter data and tag name are transferred with them to the internal storage, so that next time they are pasted all the appropriate data are available.

D 5.8.5 Paste

The following possibilities are available for inserting elements previously copied or cut:


Context menu → *Paste*
or
Edit → *Paste*


Press the key combination CTRL-V or SHIFT-INS

After pasting, a surrounding rectangle with a dashed border appears at the position in which the block was previously cut or copied.
Pasted blocks are given a new processing sequence number and assigned the status **incorrect**. Their assigned parameters are pasted in with them. If more than one block is pasted at once, their processing sequence relative to one another is preserved.

D 5.8.6 Move block

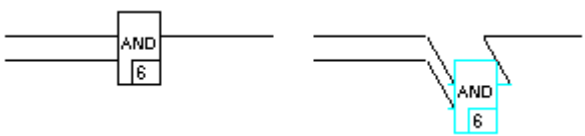
The following possibilities are available for moving a block:

Click on a selected element and hold the mouse button down. The rectangle will then appear around the selected block, and the block can now be moved by moving the mouse. When the destination position is reached, the left mouse button is released. If it is not possible to paste at the destination position, this is signaled by a warning tone, and the surrounding rectangle remains active.

If the cursor is moved into the rectangle that appears after a block is pasted, it changes into a cross with one arrow for each horizontal and vertical direction of movement. The block can then be moved by holding the left mouse button down and moving the mouse. At the destination position the mouse button is released. If it is not possible to paste at the destination position, this is signaled by a warning tone, and the surrounding rectangle remains active.

The cursor is moved over a selected element or into the rectangle that is displayed after a block is pasted, and then the space bar is pressed. This changes the cursor into a cross with one arrow for each horizontal and vertical direction of movement. The block can then be moved using the cursor keys or mouse. The block is pasted at the destination position by pressing the space bar. If it is not possible to paste at the destination position, this is signaled by a warning tone, and the surrounding rectangle remains active.

The selected elements are moved to a new position, while the element contours remain visible. All signal flow lines concerned are interrupted by the moving action. They must be corrected later. Until then, they are displayed as a "rubber band". Blocks whose mandatory terminals are not supplied with data retain the incorrect state for this period. Any parameters already configured are preserved.



Displaying a block before and after move

D 5.8.7 Import block

Context menu → *Edit* → *Import block*
or
Edit → *Import block*

A *File Open* dialog box appears, containing a list of all the files that have been generated through Export Block with the FBD editor. Once a file has been selected, the block is imported, and the rectangle surrounding the block appears. This must then be moved to a suitable position.

Imported variables that are not yet included in the variable list are displayed in red.

D 5.8.8 Export block

Context menu → *Edit* → *Export block*
or
Edit → *Export block*

A *File Save* dialog box appears, containing a list of all previously exported files in the most recently selected export directory.
Tag names are not exported.

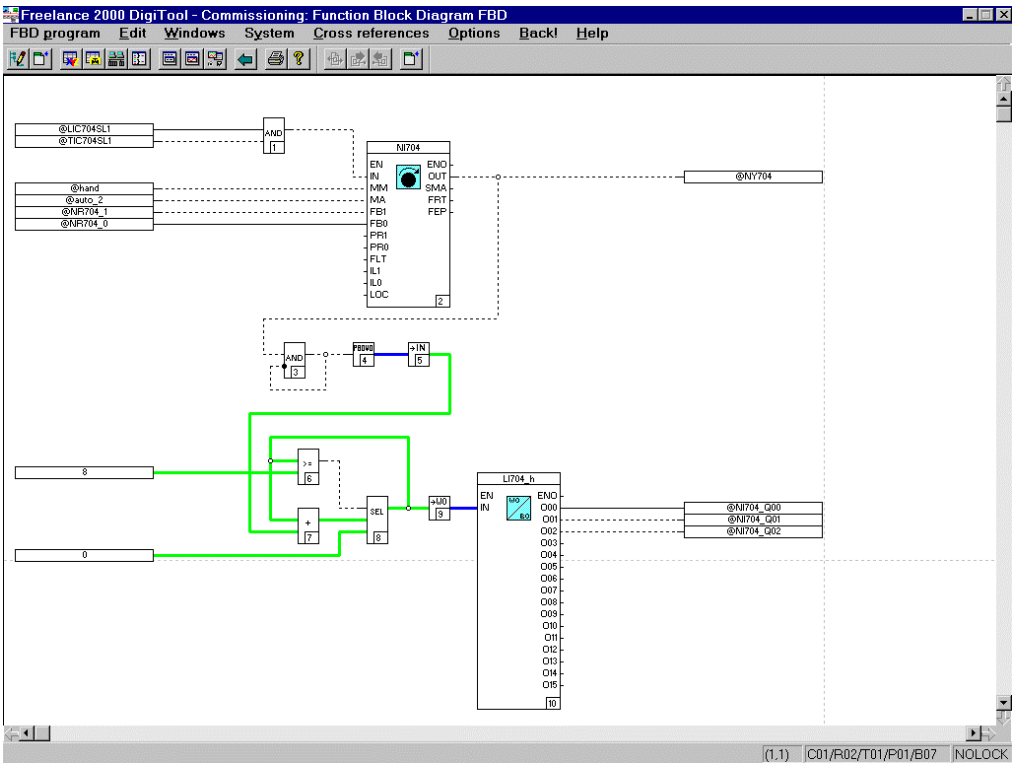
D 5.9 Undoing a procedural step

Edit → *Undo*
or
Context menu → *Undo*

This function enables one to undo the last action performed. Nonetheless, the program state continues to be **incorrect** until the next check.

D 6 Commisioning the Function block diagram (FBD)

On commissioning the FBD language the program is displayed in the same way as in configuration except that in commissioning mode the program cannot be modified structurally. .



tg008us.bmp

Individual function blocks can be selected and parameters set for them. Operating modes can also be called up and modified from commissioning mode
Thereafter, certain program test functions are available to whoever is commissioning the system.

Boolean values (binary values) are initially displayed directly with their logical state of 1 or 0.

logical 1 ——————————— true
logical 0 ---------- false

When the variables or terminals of a block are overrun, the current calculated values should be read.

After this, values within a cycle can be defined only once. Function block pins can also be defined to analog or binary values.

Input pins of function blocks which are not loaded can thus be assigned permanent values. This can be difficult notice later and should therefore be used with caution.

Click right mouse button on variable or function block pin → Input values → OK

The writing of a value should not be confused with forcing in the I/O module. The value written can be overwritten by the program in the next cycle.

D 7 Variable List and Tag List

D 7.1 Entries into the variable list

System → Variable list

Changeover is effected to the variable list. For a description, see **chapter Variables.**

The variable list contains all the variables used in the system. A variable can be selected in the list and entered into the program.

D 7.2 Entries into the tag list

System → Tag list

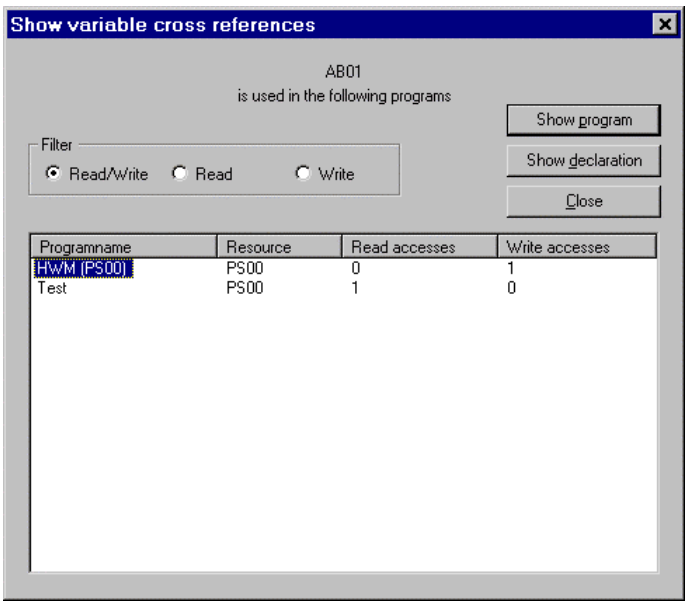
The tag list is called. For a description, see **chapter Tags.** It features a list of all tag names allotted in the system.

If tag names assigned for function block calls in the FBD program are deleted in the tag list, the entries in the corresponding parameter definition masks are blank after return from the tag list and must be reentered.

D 8 Cross References

The cross references can be selected directly from the FBD program, as follows:

- Select a variable, I/O component or tag,
- Choose menu option *Cross references* or press function key F5.



tl001us.bmp

The window offers several sorting and filtering possibilities, and settings made in this window are stored.
In contrast to the variables, for the tags no read or write access is defined.

SHOW PROGRAM	Calling a program with pre-selection of this variable or calling the module to which I/O component the variable is allocated.
SHOW DECLARATION	Step change to the corresponding I/O component in the I/O editor if a variable has been allocated to an I/O component.
Filter	A filter lets you display only those variables which are edited in read (sink) or write (source) mode in the individual programs.

After activation it is possible to branch to the programs listed as cross references. If the menu option *Back!* is chosen in the program thus reached, then control jumps back to the editor from which the branch was initiated.

Show next / previous cross reference

- Select a variable , menu: Cross references → Find next or Find previous

The next or previous use of the selected variable within the current program is displayed

D 9 General Processing Functions

D 9.1 Save the program

FBD program → Save

The program is saved without quitting the program. Incorrect programs can also be saved and completed when desired.

Program modification is not effective if the project has not been saved on closing the project or previously in the project tree.

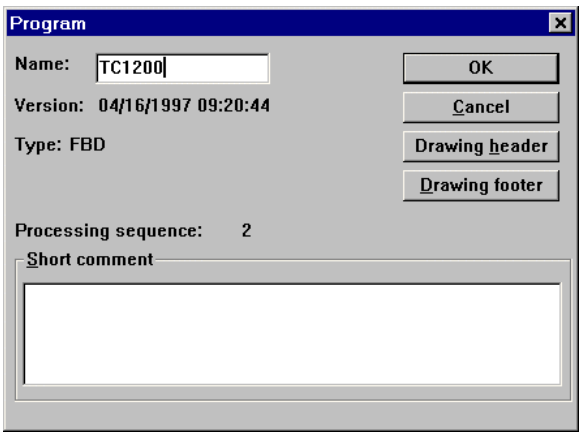
D 9.2 Document the program

FBD program → Documentation

Changeover is effected from the program to the document management, where the project documentation is user-defined and output. For a description see **Engineering Manual, System Configuration, Documentation**.

D 9.3 Program header

FBD program → Header



tl026gr.bmp

A program-specific short comment for the header line of the program documentation can be entered or edited.

DRAWING FOOTER / HEADER See **Engineering Manual, System Configuration, Documentation**

D 9.4 Edit program comment

FBD program → Comment

A program-specific long comment can be edited here for describing the functional capabilities.
For a description see **Engineering Manual, System Configuration, Project Manager, Edit comment**.

D 9.5 Back!

Back

The FBD program is exited and the application from which the last changeover was effected is called (single-step return)

D 9.6 Exit FBD program

FBD program → Exit

The FBD program is exited and the project tree called.

D 9.7 Generating hardcopy

Options → Hardcopy
→ Confirm printer setting in the Windows Print menu mask (Setup)

The screen contents are output on the screen.

D 9.8 Check of program elements

Select program element
→ FBD program → Check

All function-relevant entries are checked for syntactical and contextual correctness. Any errors and warnings are shown in the error list. If errors are detected by the check, the processing state of the program element features the incorrect state.

Any newly entered, copied or moved program elements have the incorrect state.

D 9.9 Delete an FBD program

Select Project tree → Select program → *Edit* → *Delete*.

The variables and tag names are preserved in other programs and in the variable/tag list and can be reassigned.

D 9.10 Copy and paste an FBD program

Select Project tree → Select program to be copied → *Edit* → *Copy* or Ctrl-C
→ Select position to which program is to be copied
→ *Edit* → *Paste* or Ctrl-V
→ Depending on position selected, select above, below or level.
→ Assign program name.

The program is copied and assigned under a new, unambiguous name to a program list of the project.

The respective configuration, incl. program header and program comment is copied.
The tag names of the function blocks are not copied.

The copied program is designated incorrect and is allotted the date and time of copying as version code.

D 9.11 Link programs

Programs are linked by variables to one another or to the input/output modules.

E Instruction List (IL)

Line	Label	Op.	Operand	()		Comment
0001		LD	@Onoff_Cnt			
0002		ADD	1			Accumulator
0003		ST	@Onoff_Cnt			
0004		LD	@Onoff_Cnt			
0005		LT	20			
0006		JMPC	L999			
0007		LD	@Onoff_Var			
0008		NEG				
0009		ST	@Onoff_Var			
0010		LD	0			
0011		ST	@Onoff_Cnt			
0012	L999					
$						

Contents

E 1 General Description - Instruction List

Instruction List (IL) is an IEC-61131-3 compliant line-oriented programming language. The program instructions have operators which act upon an explicit operand and the accumulator to give an intermediate result which is then itself saved in the accumulator.

All the functions and function blocks in DigiTool are available in IL. The functional scope of the functions is, for the most part, covered by IL operators. When a function block is selected from the menu, however, a CAL operator and a list of input and output is inserted. The programmer should then fill in this list, assigning signals by name. Parameters are assigned using the same masks as in function block or ladder diagram.

IL is considerably more flexible than FBD, because it provides for jump and loop functions with the corresponding program tags (labels).

The signal flow is not as easy to follow or document as in FBD; therefore provisions have been made to allow adding a comment to any instruction line.

The program instructions for IL can be selected from a list by pressing F2. Program flow automatically follows the order of the instructions (from top to bottom). The sequence can only be changed by intentionally inserting a **jump, return** and **loop operator**.

IL programs can be up to 1000 lines in length.

E 1.1 Creating a new IL program

IL programs can be created or called for editing from an active program list, from the project pool or from an SFC program (establishing the transition conditions or the actions for a step). See also **Engineering Manual, System Configuration, Project Tree** and **chapter Sequence Function Chart**.

An IL program is created from the project tree using the following steps, for example:

→ *Project tree* → Select insert position in project tree
→ *Edit* → *Insert above, Insert below* or *Insert next level*
→ IL program from "Object selection"
→ Enter program name and short comment if necessary

Each new IL program has a blank instruction list, the editing state incorrect and the creation date as the version identification. The name of the program listing is preset as the program name, but this can easily be changed.

E 1.2 Calling the editor

As soon as the IL program exists, it can be called for editing as follows:

→ **Project tree** → Move cursor to program name and double-click on left.
or
→ Select program by clicking on the left,
→ *Edit* → *Program*

New lines may now be entered or entries changed in the instruction list displayed.

E 2 Interface of the IL Program

E 2.1 Design of the IL configuration interface

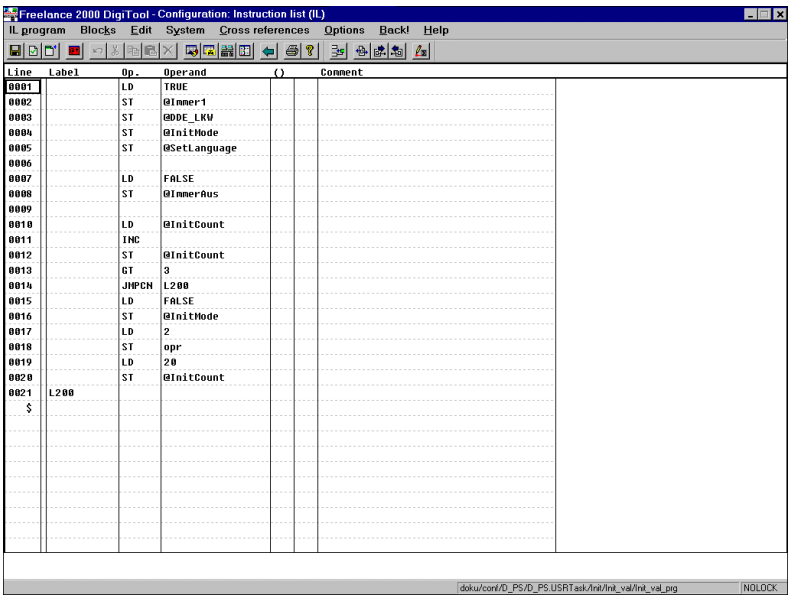
The configuration interface of the IL editor consists of the following elements:

Menu line

Column headings

Report lines

Status line



di0401uk.bmp

└ Mark

└ Commissioning field

└ Parenthesis depth

Line — The line number is allocated automatically in consecutive sequence from 1 to 1000. When blank lines or command lines are inserted, the line numbers of subsequent command lines are automatically displaced by the number of lines inserted.

Mark — All the lines belonging to a function block are labeled here in color unless the mandatory parameters contained therein are fully assigned. Once they are fully assigned, these fields become gray.

Label — Jump labels L001 up to L999 (label), which act as transfer addresses for jump operators, are entered in this column. The entry is not tied to any sequence. It is nevertheless recommended to aim for an ascending sequence, but to use only full figures of tens at first, so as to be able to insert further jump labels later in monotone sequence. The monotone sequence makes searching easier in longer program listings.

Operator (Op.)

Once a field has been selected in this column, the operator can be entered by key input or by selection from a menu, which can be called using F2. Depending on the operator type, a (suitable) argument should then be specified if necessary in the adjacent column (see page **E-11**, **Acceptable data types for IL operators and functions**).
In the case of function block, this field is assigned automatically following block selection (see page **E-20**, **Inserting function blocks into an IL program**).

Operand

In the case of jump operators, the jump label should be entered here, whereas logical operators require a constant or a variable as an argument.
Special conditions apply here also for function blocks (see page **E-20**, **Inserting function blocks into an IL program**).

Parenthesis depth ()

When parenthesizing logical operators, a number 1 ... 8 appears here, which indicates the depth of parenthesis (see page **E-21**, **Changing the number of inputs to function blocks**)

Commissioning field

If the program is commissioned and processing is in progress, a T for logical 1 (TRUE) or an F for logical 0 (FALSE) is shown here when the contents of the accumulator are Boolean.

Comment

Explanations can be entered here to aid understanding of the program run, e.g. on the meaning of variables, the function of the program section or the function block called.

Status line

The state line indicates the name of the program which is being edited and name of the user.

E 2.2 Menu structure

IL program
Save
Documentation
Check
Header
Comment
Exit

Blocks
Analog
Binary
Constants
Converter
Acquisition
Arithmetic
Converter
Standard
Open loop control
Modbus master
Modbus slave
Monitors
System functions
TCP/IP send and receive blocks
User function blocks
User menu

Edit
Undo
Insert line
Field
Delete field
Change data type
Parameters
Change number of inputs
Zoom to user FB
Select variable
Toggle use of process image
Cut
Copy
Paste
Delete
Export block
Import block

System
Variable list
Tag list
Hardware structure
Structured data types

Cross references
Cross references
Find next
Find previous

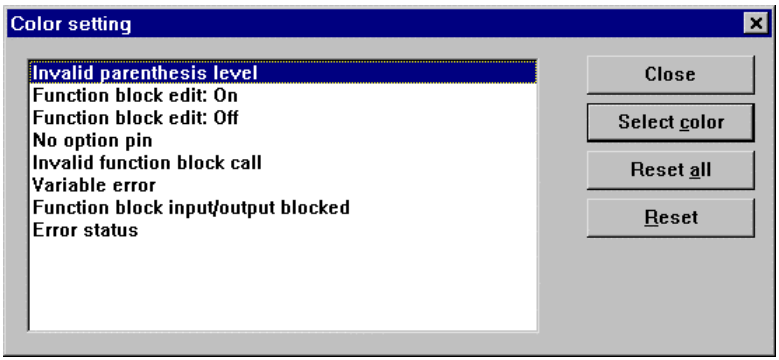
Options
Version
Hard copy
Define user menu
Colors
Save column settings

Back!

Help
Contents
Overview
Use help
About

E 2.3 Changing the color display

→ *Options* → *Colors*
→ Select object of which the color is to be changed.
for example, the color for labeling invalid parenthesis levels
→ select desired color

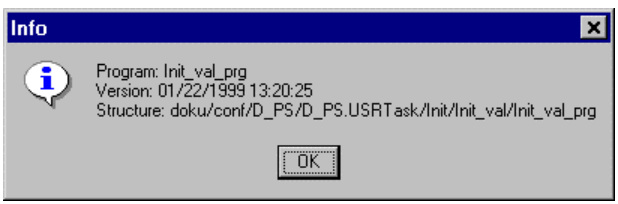


di0412uk.bmp

E 2.4 Showing the version information

→ *Options* → *Version*

The program name, date of last program change (version) and allocation of the program to the project, resource, task and program listing are shown. The program allocation can be shown as long or short text. The setting for this is made in the project tree.



di0415uk.bmp

E 3 Editing an IL Program

Due to the list structure of the editing interface, the functions outlined in the description Variable and Tag list apply by analogy. All operating steps e.g. for selecting fields, labeling, deleting, moving or copying blocks, are described there and work in exactly the same way with IL.

Also see **chapter Variables** and **chapter Tags**

E 3.1 Acceptable data types for IL operators and functions

The data types which are possible in DigiTool can be divided into the classes bit strings (any_bit), integer numbers (any_int), floating-point numbers (real) and special formats for time and date.
Bit strings and integer numbers are also defined in various data widths and/or with or without a sign. The 11 formats currently available are entered in the following table as columns. The table provides information in the form of a matrix showing which IL operators can process which data types:

	any bit				any int					time / date	
	BOOL	BYTE	WORD	DWORD	INT	DINT	UINT	UDINT	REAL	TIME	DT
LD, ST	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
LDN, STN	✓	-	-	-	-	-	-	-	-	-	-
AND, OR, XOR	✓	✓	✓	✓	-	-	-	-	-	-	-
ANDN, ORN, XORN	✓	✓	✓	✓	-	-	-	-	-	-	-
S, R	✓	-	-	-	-	-	-	-	-	-	-
NEG	✓	✓	✓	✓	✓	✓	-	-	✓	-	-
DEC, INC	-	-	-	-	✓	✓	✓	✓	-	-	-
SL, SR, RL, RR	-	✓	✓	✓	-	-	-	-	-	-	-
EQ, GE, GT, LE, LT, NE	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
ADD, SUB	-	-	-	-	✓	✓	✓	✓	✓	1)	1)
MUL, DIV, MOD	-	-	-	-	✓	✓	✓	✓	✓	2)	2)

[1]) acceptable: ‹DT› +/- ‹TIME› = ‹DT›
[2]) acceptable: ‹TIME› */: ‹INT› = ‹TIME›

If blocks are used in Instruction List, the acceptable data types are dictated by the block type. In the case of blocks for different data formats (see table below), a menu window is opened in which the data type is selected.

Blocks with several data types:

	any bit				any int					time / date	
	BOOL	BYTE	WORD	DWORD	INT	DINT	UINT	UDINT	REAL	TIME	DT
ABS	-	-	-	-	✓	✓	-	-	✓	-	-
AVG	-	-	-	-	✓	✓	-	✓	✓	✓	-
MIN, MAX	-	-	-	-	✓	✓	-	✓	✓	-	✓
MUX	✓	-	-	-	✓	✓	-	✓	✓	-	✓
SEL	✓	-	-	-	✓	✓	-	✓	✓	-	✓
TRUNC	-	-	-	-	✓	✓	-	✓	✓	-	-

The conversion block *TO*, which converts a variable of one data type into a variable of another data type, constitutes a special feature here. Conversion is implemented at present for the following data types
See also **Engineering Reference Manual, Functions and Function Blocks, Converter Blocks**.

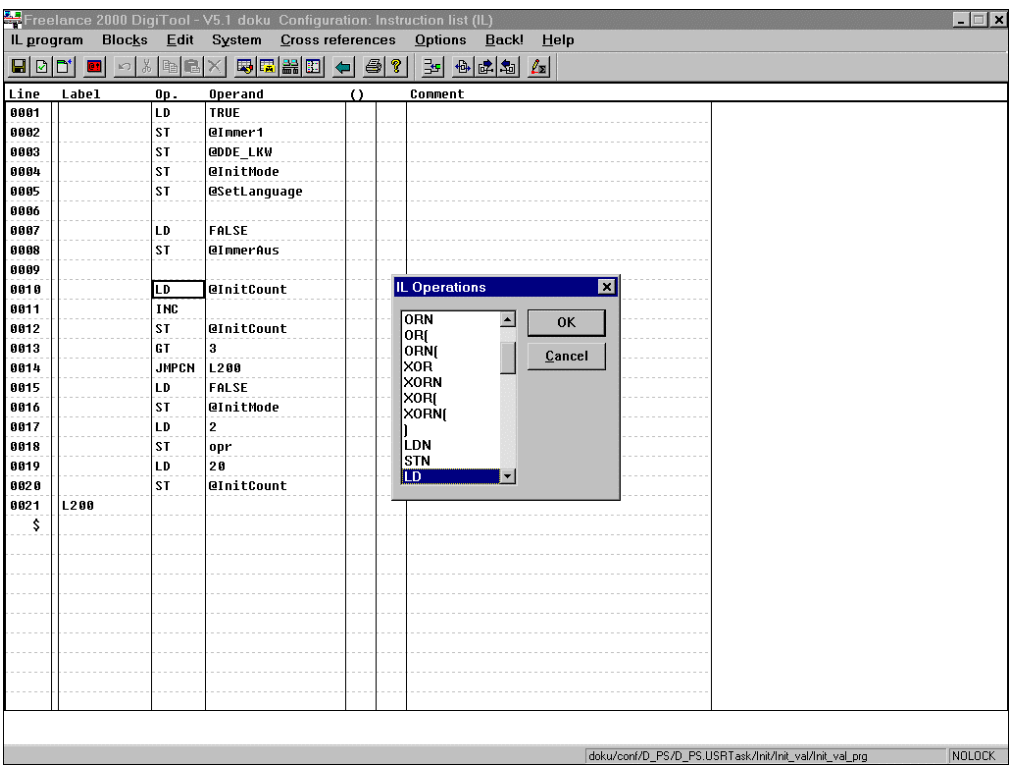
Input \ Output	INT	UINT	DINT	UDINT	BYTE	WORD	DWORD	BOOL	REAL	TIME	DT
INT		TO	TO	–	–	TO	–	–	TO	–	–
UINT	TO		–	TO	–	TO	–	–	TO	–	–
DINT	TO	–		TO	–	–	TO	–	TO	TO	–
UDINT	–	TO	TO		–	–	TO	–	TO	TO	–
BYTE	–	–	–	–		PA	PA	EX	–	–	–
WORD	TO	TO	–	–	EX		PA	EX	–	–	–
DWORD	–	–	TO	TO	EX	EX		EX	TO	TO	–
BOOL	–	–	–	–	PA	PA	PA		–	–	–
REAL	TO	TO	TO	TO	–	–	TO	–		–	–
TIME	–	–	TO	TO	–	–	TO	–	–		–
DT	–	–	–	–	–	–	–	–	–	–	

Blocks: TO = *_TO_xx
PA = PACK
EX = EXTRACT

E 3.1.1 Entering constants

Constant numerical values can be input according to data type with or without a sign in binary, octal, decimal or hexadecimal format. Floating-point numbers should always be input with the decimal point, even if accompanied by an exponent.
To differentiate them from decimal numbers, binary, octal and hexadecimal numbers are preceded by a suitable identification character (2#, 8# or 16#, see below).
The possible data types are described in **chapter Variables**

E 3.2 Calling IL operators



di0403uk.bmp

The structure of IL programs is adapted to that of assembler programs of simple microprocessors with an accumulator. Constants or variables are loaded into this "accumulator", combined with other quantities, transformed and saved in a target quantity.
Operators are the basic elements of the instruction set. They can be subdivided into the groups Logic, Basic Arithmetic, Comparators, Shift Instructions for Bit Strings and load, save and other organizational instructions.

Once an operator field has been selected, the list of operator types currently available can be called using the F2 key and the desired operator selected by means of the cursor and return keys. The shorthand symbol for the operator may also be entered directly, bypassing the selection menu (RETURN key in operator field and enter letter; completion again by means of RETURN key). To separate program sections from one another by a blank line, the line following the desired point of separation is selected, Edit is called, the menu item *Insert line* is selected and confirmed by pressing the ENTER key.

E 3.2.1 Loading and saving data

All data and signal types are loaded into the accumulator using the operator **LD.** In the case of Boolean data/signals the operator **LDN** may also be used, which loads the input quantity into the accumulator in inverted form. The corresponding operators for storing the accumulator contents are **ST** or **STN**.
Since a storage operator does not change the accumulator, it can be used several times in succession to distribute the same contents to various outputs. The output variables must be of the same type as the accumulator contents, otherwise an appropriate type corruption message giving the pertinent line number will be generated during the plausibility check.
Boolean output quantities and variables may also be set to logical 1 by means of the operator S (=set) and to logical 0 using **R** (= reset), if the accumulator contents include a logical 1. The argument variable is thereby treated like a flip-flop.

E 3.2.2 Logic operations

Boolean and other bit string quantities can be combined with one another using the operators **OR**, **AND**, **XOR** (= exclusive or). These logical operators can be combined with the supplements **"N"** (= negated) or/and **"("** (= left parenthesis). A complete list of all IL operators is featured on **page E-19, Overview of IL operators**
The table below provides information on the meaning of the individual logic operations. In-depth treatises on the theory of logic operations are to be found in specialist literature on the subject.

Function	**AND**		**ANDN**		**OR**		**ORN**		**XOR**		**XORN**	
Argument	0	1	0	1	0	1	0	1	0	1	0	1
Accu = 0	0	0	0	0	0	1	1	0	0	1	1	0
Accu = 1	0	1	1	0	1	1	1	1	1	0	0	1

Explanation:
The two states of the accumulator contents to date (line: accumulator = 0 or 1) combined with the two states of the argument (column: 0 or 1) supply the four possible results in the accumulator (at the point of intersection of the line/column).

Example: Accumulator = 1 **XORN** with argument 0 gives the accumulator result = 0).

E 3.2.3 Logical operators with parentheses

The supplement "left parenthesis" named in the previous section, together with the operator ")" (right parenthesis) makes it possible to convert even complex logic operations into corresponding IL line sequences. In principle, all operations can be formulated even without parentheses if intermediate results are filed in flags and reloaded later. However, this calls for more instruction lines, and clarity is reduced. Nevertheless, the number of lines can be reduced markedly by the skillful use of partial results in the accumulator. It is often possible to avoid storing intermediate quantities simply by re-sorting the operations.
Parentheses may be nested up to a depth of 8 levels. The respective parenthesis depth is shown in Instruction List in the 6th column. Red question marks appear in this column only if the 8th level is exceeded. The parenthesis depth shown must be brought back down to 0 again in subsequent lines using right parenthesis operators.

IL without parentheses with all intermediate variables		IL without parentheses, intermediate variables reduced		IL with parentheses [operation converted]	
LD	bool1	LD	bool1	LD	bool1
OR	bool2	OR	bool2	OR	bool2
ST	z1	ST	z1	AND(	bool3
LD	bool3	LD	bool3	OR	bool4
OR	bool4	OR	bool4	)	
ST	z2	AND	z1	OR(	bool5
LD	bool5	ST	z7	OR	bool6
OR	bool6	LD	bool5	AND(	bool7
ST	z3	OR	bool6	OR	bool8
LD	bool7	ST	z3	)	
OR	bool8	LD	bool7	)	
ST	z4	OR	bool8	ORN(	bool5
LDN	bool5	AND	z3	AND	bool6
ORN	bool6	ST	z8	OR(	bool7
ST	z5	LDN	bool5	AND	bool8
LDN	bool7	ORN	bool6	)	
ORN	bool8	ST	z5	)	
ST	z6	LDN	bool7	ST	boolX
LD	z1	ORN	bool8		
AND	z2	AND	z5		
ST	z7	OR	z8		
LD	z3	OR	z7		
AND	z4	ST	boolX		
ST	z8				
LD	z5				
AND	z6				
ST	z9				
LD	z7				
OR	z8				
OR	z9				
ST	boolX				

E 3.2.4 Relational operators

Two quantities of the same data type (previous accumulator contents and argument) are compared with one another using the relational operators EQ ... LE and the result saved in the accumulator as a Boolean variable (see page **E-20**, **Inserting function blocks into an IL program**). The relation functions can also be called as blocks, but are not distinguished by this from the operators.

E 3.2.5 Numerical operations

The operators ADD, SUB, MULT, DIV, MOD can be used to combine two quantities (accumulator and argument) of the same data type (exceptions in the case of TIME and DT data type, see page **E-11**) numerically. The result is then available in the accumulator for storage or for further operations. Numerical operations can also be called as blocks.
However, the addition and multiplication blocks are distinguished from the corresponding operators by the fact that they can be used for multiple inputs (see page **E-20**, **Inserting function blocks into an IL program**).

E 3.2.6 Shift operators

The shift operators (SL, SR, RL, RR) can only be used for bit string formats, i.e. for Byte, Word and DWord. They do not require an argument. The bit string is moved by one space to the left or right respectively. In the case of SL, SR, the space which then becomes free is filled by a 0, while in the case of RL, RR the bit pushed out of the format is reinserted at the other end. Example: RR(10111101) gives 11011110.
The shift instructions called as blocks are not distinguished from the accompanying operators.

E 3.2.7 Loop operators

In offering the opportunity to incorporate repeat loops into programs, the IL language differs markedly from FBD. One of the loop start operators WLC, RPC or WLNZ respectively appears at the start of the loop, followed by the "loop core", which is to be executed several times, consisting of load, processing and storage operators as well as block calls. At the end of this part, the loop terminate operator LPE is inserted.

Loop starting instructions have the following meaning:

WLC **W**hi**L**e **C**ondition skips the loop if the accumulator is not logical 1.

RPC **R**e**P**eat on **C**ondition checks the accumulator only at the end of the loop (in the line with LPE). If it is logical 1, the loop is executed once more.

WLNZ **W**hi**L**e **N**ot **Z**ero checks a counter defined by "Argument" with UDINT format (at the beginning of the loop). If it is zero, the loop is aborted, otherwise it is executed.

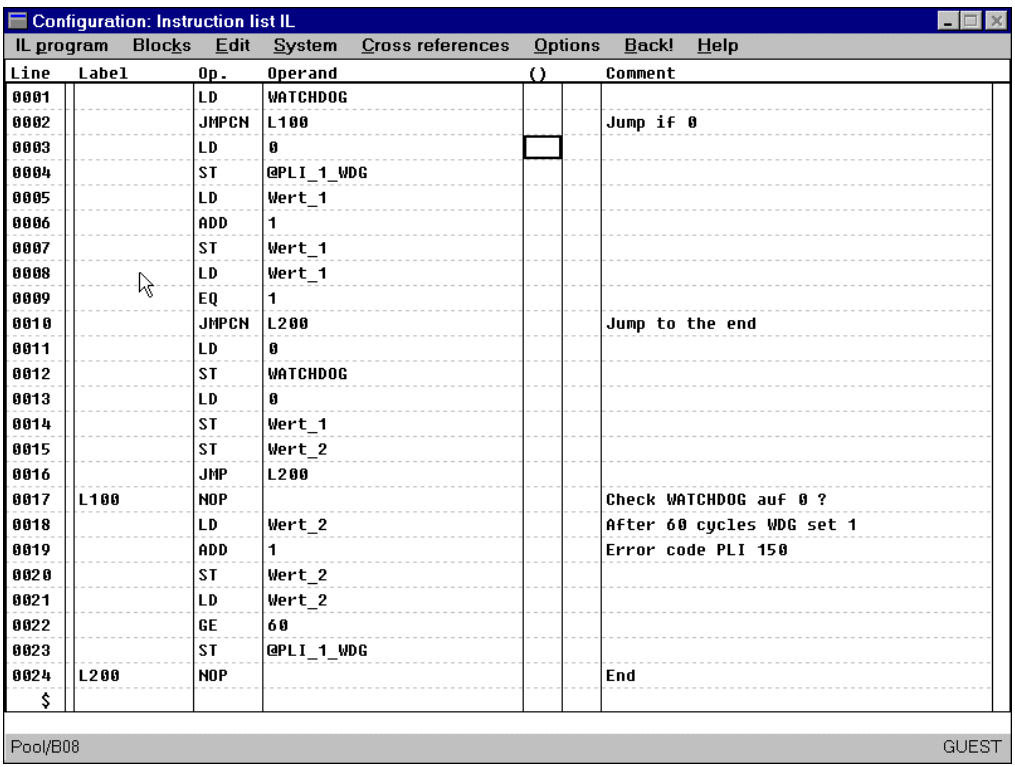
All three types of loop can degenerate into endless loops as a result of poor programming. It is the programmer's responsibility to prevent this from happening.

Example of an IL program with loop operator:
The program signals a logical 1 following TempFlr if at least one of the temperatures Temp1 ... Temp7 is greater than the fixed value 70 °C.

	LD	MaxKnl	maximum number of channels to be monitored
	ST	UDZLR	save → UDZLR
	GT	7	if greater than 7
	RETC		terminate program
	LD	1	initial value 1
	ST	ZLR	save → ZLR
	WLNZ	UDZLR	process loop up to LPE, if UDZLR > 0
	LD	ZLR	channel counter as selection criterion for multiplexer
	MUX	Temp1	Channel 1
	'	Temp2	Channel 2
	'	Temp3	Channel 3
	'	Temp4	Channel 4
	'	Temp5	Channel 5
	'	Temp6	Channel 6
	'	Temp7	Channel 7
	GT	70.0	if selected temperature greater than 70.0 °C,
	JMPC	L030	then jump (with accumulator = 1) → L030
	LD	ZLR	
	INC		increment selection channel ZLR by 1
	ST	ZLR	
	LPE		loop end
	LD	FALSE	since no temperature greater than 70 °C, load logical 0
L030:	ST	TempFlr	save accumulator contents → TempFlr
	RET		program end

E 3.2.8 Jumps and program calls

Using the jump operators JMP, JMPC, JMPCN the program can be continued at the point named in the argument: i.e. the lines lying in between are skipped. The jump destination must lie **below** the line containing the jump operator. It should be entered in the destination line by means of an identifier in the form L001 ... L999.



Line	Label	Op.	Operand	()	Comment
0001		LD	WATCHDOG		
0002		JMPCN	L100		Jump if 0
0003		LD	0		
0004		ST	@PLI_1_WDG		
0005		LD	Wert_1		
0006		ADD	1		
0007		ST	Wert_1		
0008		LD	Wert_1		
0009		EQ	1		
0010		JMPCN	L200		Jump to the end
0011		LD	0		
0012		ST	WATCHDOG		
0013		LD	0		
0014		ST	Wert_1		
0015		ST	Wert_2		
0016		JMP	L200		
0017	L100	NOP			Check WATCHDOG auf 0 ?
0018		LD	Wert_2		After 60 cycles WDG set 1
0019		ADD	1		Error code PLI 150
0020		ST	Wert_2		
0021		LD	Wert_2		
0022		GE	60		
0023		ST	@PLI_1_WDG		
0024	L200	NOP			End
$					

di0409uk.bmp

The jump is always executed if JMP is specified. In the case of JMPC, it is only executed if the accumulator = logical 1, and for JPMCN only if the accumulator = logical 0.

Program calls are not available at present. The following operators are envisaged for these:

CAL	unconditional call
CALC	call only when accumulator = logical 1
CALNC	call only when accumulator = logical 0

Further details will be provided once they have been implemented.

E 3.2.9 Overview of IL operators

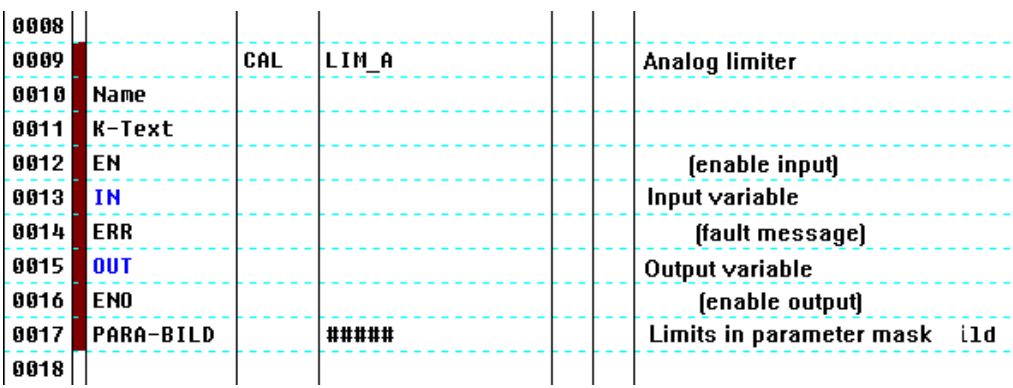
AND	Accumulator AND argument to accumulator **(= Accumulator)**
ANDN	Accumulator AND (argument negated)
AND(	Accumulator AND left parenthesis
ANDN(	Accumulator AND negated, left parenthesis
OR	Accumulator OR argument to accumulator
ORN	Accumulator OR (argument negated)
OR(	Accumulator OR left parenthesis
ORN(	Accumulator OR negated, left parenthesis
XOR	Accumulator EXOR argument to accumulator
XORN	Accumulator EXOR (argument negated)
XOR(	Accumulator EXOR left parenthesis
XORN(	Accumulator EXOR negated, left parenthesis
)	Right parenthesis
LDN	Load argument in inverted form to accumulator
STN	Save accumulator in inverted form to argument
LD	Load argument to accumulator
ST	Save accumulator to argument
S	Set argument variable to logical 1 if accumulator = 1
R	Set argument variable to logical 0 ("Reset") if accumulator = 1
EQ	If accumulator is equal to argument, logical 1 to accumulator, otherwise logical 0.
NE	If accumulator is not equal to argument, logical 1 to accumulator, otherwise logical 0
GT	If accumulator is greater than argument, logical 1 to accumulator, otherwise logical 0
GE	If accumulator is greater than or equal to argument, logical 1 to accumulator, otherwise logical 0
LT	If accumulator is less than argument, logical 1 to accumulator, otherwise logical 0
LE	If accumulator is less than or equal to argument, logical 1 to accumulator, otherwise logical 0
ADD	Accumulator plus argument to accumulator
SUB	Accumulator minus argument to accumulator
MUL	Accumulator times argument to accumulator
DIV	Accumulator divided by argument to accumulator
MOD	Accumulator divided by argument, remainder to accumulator
NEG	Negate accumulator
INC	Increment accumulator (+1)
DEC	Decrement accumulator (-1)
NOP	No operation
SL	Move bit string in accumulator 1x to left, 0 moves up
SR	Move bit string in accumulator 1x to right, 0 moves up
RL	Rotate bit string in accumulator 1x to left
RR	Rotate bit string in accumulator 1x to right
WLC	If accumulator = logical 1, execute the following lines as far as LPE

RPC	As for WLC, but loop is executed at least once
WLNZ	If the integer counter named by argument is not zero, execute the lines as far as LPE. With every loop the counter will be decremented by 1.
LPE	End of a repeat loop
JMP	Jump to label indicated in argument field unconditionally
JMPC	Jump if accumulator = logical 1
JMPCN	Jump if accumulator = logical 0
RET	Return from program (sub-program) unconditionally
RETC	Return if accumulator = logical 1
RETCN	Return if accumulator = logical 0

E 3.3 Inserting function blocks into an IL program

All the function blocks available in FBD programming can also be called in IL by way of the menu item *Blocks*.

The function blocks are "named" blocks, i.e. they are entered into the instruction list using a CAL operator and receive a name, a comment and a parameters display. When they are called, a fixed block of IL lines is inserted into IL ahead of the selected list position. A line is reserved in each case for all inputs and outputs. All lines of the block except the CAL line contain an identifier text, which identifies the respective signal. The identifiers for necessary inputs/outputs (mandatory parameters) are highlighted in color.
Certain argument fields have a gray background if the relevant input has already been occupied in the parameters display by a constant quantity. Column 2 is marked in color if all mandatory parameters of the block have not yet been properly entered or the block has been taken out of processing, otherwise the color marking is gray. It is possible to tell from this marking that a block is being used, even in the case of instructions which are available as an operator and as a block.



di0410uk.bmp

The parameters display which goes with the named block is selected as follows:

E 3.3.1 Defining parameters for function blocks

Double click with left mouse button on the field PARADISP.

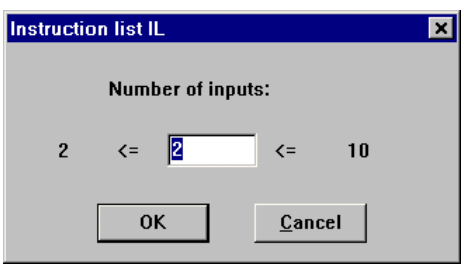
The parameters displays are the same as for FBD programming. Detailed information on parameter assignment is provided in **the Engineering Reference Manual, Functions and Function Blocks.**

It is possible to quit the parameters display at any time by pressing the ESC key.
The comment field in the last line of the block initially contains a row of 5 hash signs (#####). This marking indicates that the block has not yet been checked successfully for plausibility. Following the plausibility check these symbols change into @@@@@. See also **chapter Function Block Diagram (FBD), Description of FBD Program Elements**.

E 3.3.2 Changing the number of inputs to function blocks

Some function blocks have a variable number of inputs (AND, OR, XOR, ADD, MUL, MUX). When such a block is called up, a dialog box pops up and the desired number of inputs within an allowable range must be filled in. The smallest reasonable value appears as the default. Note that for MUX blocks, the selection signal (INT range) must also be counted.

Instead of using function blocks with multiple inputs, the corresponding single operators can be combined appropriately to give the same result. This eliminates the limitation to 10 inputs.



di0413uk.bmp

E 3.4 Plausibility Check

If something was missed, forgotten or entered incorrectly during program input, the plausibility check supplies the formal warning and error messages, together with the line number, so that corrections can be made prior to commissioning. The program can only be downloaded into the process station and operated if a plausibility check has been accomplished without any error messages. The plausibility check is called from the menu *IL program*:

 → *IL program* → *Plausibility check*

All inputs that bear relevance to the function are checked for syntactical and contextual correctness. Any errors that are found are displayed along with warnings in an error list. If errors are detected by the plausibility check, the processing state of the program element is set to **incorrect**.

E 3.4.1 Jumping to error locations after the plausibility check

After a new plausibility list has been created via *Plausibility check* or *Plausibility check all*, any errors that were detected are displayed to the user in a list box.

- Double-clicking the left mouse button or pressing [Enter] on a marked message will execute a jump to the field causing the error.

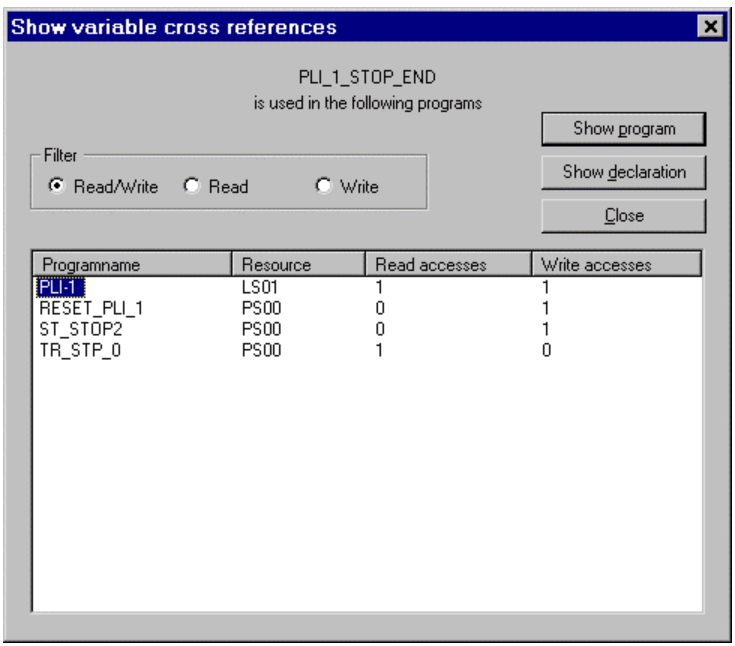
After returning to the project tree and calling *Display errors*, the next plausibility message in the list will appear selected.
The help text relating to the marked message can be called up via a [Help] button.
The destinations of these jumps are exactly the same whether the plausibility check was called up in the project tree or in the program. If a jump is performed to a program page in which a selection was available previously (only possible after running plausibility check within an editor), then the selection will be lost in the process.

See also **Engineering Manual, System Configuration, Project Tree, Plausibility Check**.

E 3.5 Selecting the Cross References

The cross-references can now be selected directly from the IL program

- Select a variable or tag,
- Menu option Cross-references or function key F5.



te001uk.bmp

The window offers several sorting and filtering possibilities. Settings made in this window will be stored.
In contrast to the variables, for the tags no read or write access is defined.

SHOW PROGRAM	Calling a program with pre-selection of this variable or calling the module to which I/O component the variable is allocated.
SHOW DECLARATION	Jump to the corresponding I/O component in the I/O editor if a variable has been allocated to an I/O component.
Filter	A filter lets you display only those variables which are edited in **read** (sink) or **write** (source) mode in the individual programs.

Once this function has been activated it is possible to **branch** to the programs listed as cross-references. If the menu option *Back!* is executed from a program branched to in this way, a jump is performed back to the original editor.

Show next / previous cross reference

- Select a variable, menu: Cross references → Find next or Find previous

The next or previous use of the selected variable within the current program is displayed

E 3.6 Compiling a User Menu

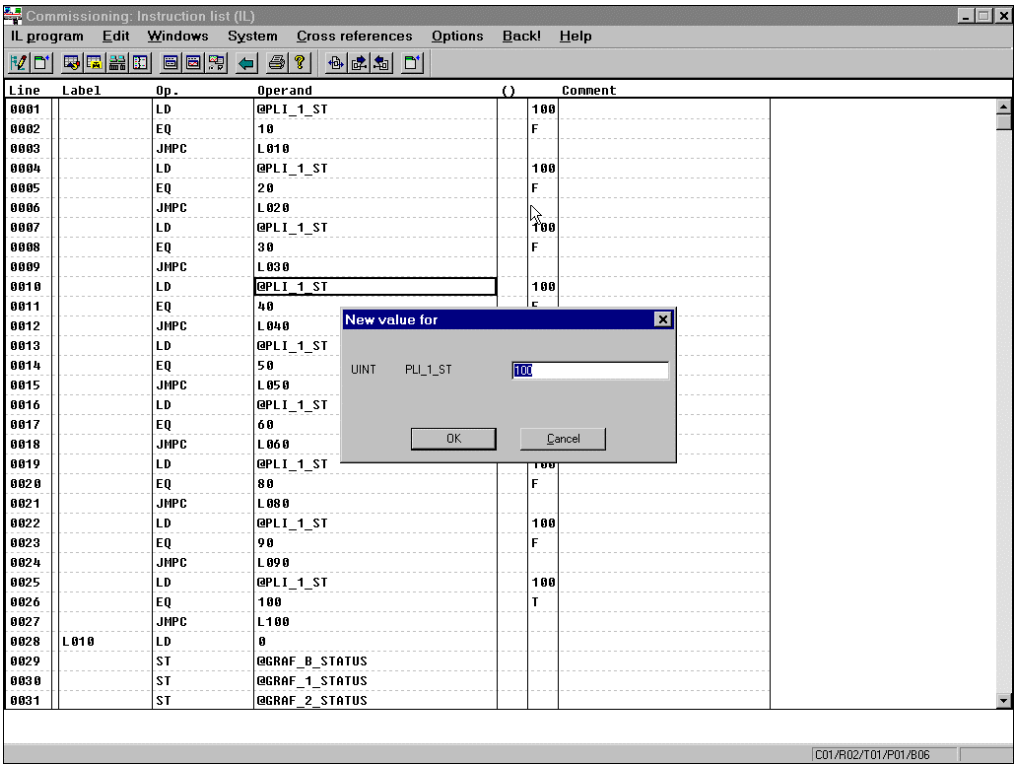
 Menu bar → *Options* → *Define user menu*

To simplify the search for frequently used blocks in the block menu, the user may compile a "preference list" from the total number of function blocks available.

Creating a user menu, see **chapter Function Block Diagram.**

E 4 Commisioning the Instruction list (IL)

The parameters associated with function blocks can be displayed and edited. In addition, the accumulator state is displayed in a column. Accumulator fields not calculated remain empty. A value can be written to a variable on a one-time basis.



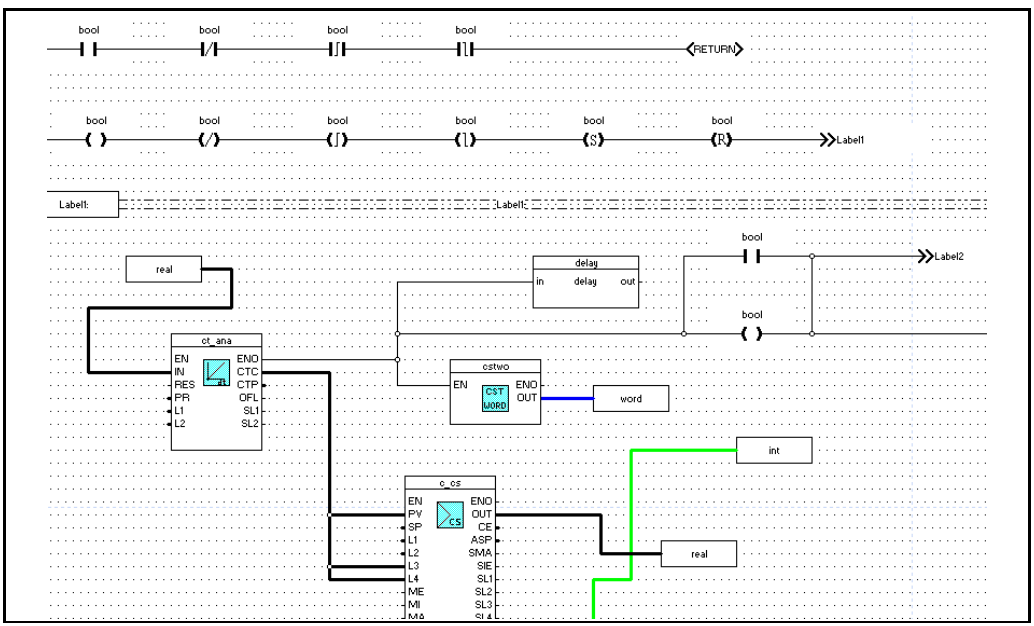
di1559uk.bmp

Values can be written once within a processing sequence.

Select variable → *Window* → *Write value* → Enter value → OK

The writing of a value should not be confused with forcing. The value written can be overwritten by the program in the next cycle.

F Ladder Diagram



Contents

F 1 General Description - Ladder Diagram Language

The Ladder Diagram is a graphically-oriented IEC 61131-3 programming language.

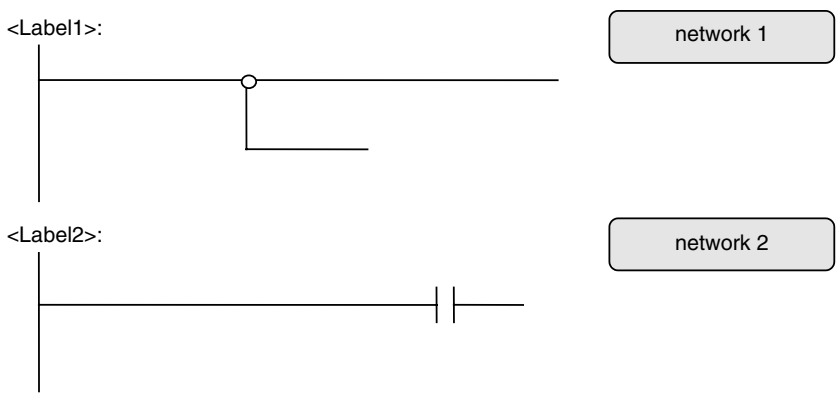
The LD language originates from the field of electromagnetic relay systems and describes the flow of current through the individual networks of the program organization units (POU) of a programmable controller.

The work area of an LD program is structured over 10 x 10 pages. An individual page can be reached by vertical and horizontal scrolling. A raster is applied to the entire work area. The breaks between pages are indicated by a dashed line. The program documentation, which is output on a page-for-page basis, gives an exact picture of what can be found on a page.

An LD network is delineated on the left and right by the so-called power rails.

An LD network comprises the following **graphic elements**:
- Connections and lines,
- Variables and constants,
- Ladders,
- Coils,
- Conditional and unconditional jumps,
- Functions and function blocks.

There can be several networks on different levels within an LD program; these will be processed passing from the top to the bottom unless any explicit jumps have been programmed in.



Rules for processing a Ladder Diagram program

An LD program is processed in accordance with the following rules:

1. No network element may be calculated until the states of the inputs have been calculated,
2. The calculation of a network element is not concluded until the states of the outputs have been calculated,
3. The calculation of a network is not concluded until all the outputs have been calculated, even if the network contains jumps - either forward or backward,
4. Networks are processed top-down.

Rule 4, however, is also dependent on the signal flow of the program, as rules 1-3 must also be obeyed. By way of illustration, the following algorithm is used for determining the processing sequence:

1. Sort all network elements from top to bottom and, within that sort, from left to right,
2. Search for the first element in which all the inputs are calculated,
3. Calculate that element,
4. If there are other uncalculated elements, go to step 2.

In contrast with the FBD language, no explicit processing sequence can be specified for the blocks, but instead it emerges from the structure of the program. Feedback messages are also not permitted, as they contravene rule 1.

As an extension of the IEC language definition, any of the structured data types may be used. Structured data types can be used for data exchange with other DigiNet S stations, using TCP/IP send and receive function blocks.

To load the programs when there is an existing connection to the process stations, the editor is switched to a special mode which enables current values to be displayed. For further details, see **Engineering Manual, System Configuration, Commissioning.**

F 1.1 Creating an LD program

The creation of an LD program is carried out in the **project tree**.

Project tree → Choose insertion position in project tree
→ Edit → *Insert above*, *Insert below* or *Insert next level*
→ LD program from “Object selection”
→ Assign a program name and, if required, a short comment

Each new LD program has an empty graphic area, no comment, an editing status of **incorrect** and a version identifier formed from its creation date. The name of the program list (PL) forms the default program name. The short comment from the program list is copied across and can be modified.

F 1.2 Calling the LD program

First, the program is selected

Project tree *Edit* → *Program*
or double-click on program

The program is displayed with its content (functions, signal flow lines etc.) and can be modified.

F 2 User Interface of the Ladder Diagram Program

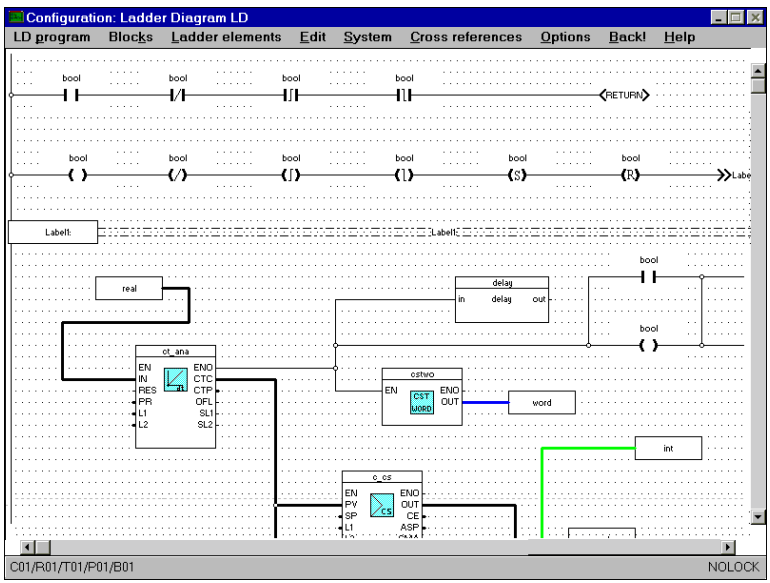
F 2.1 Structure of the configuration interface

The configuration interface of an LD program comprises:

Menu bar

Graphic area

Status line



tn001us.bmp

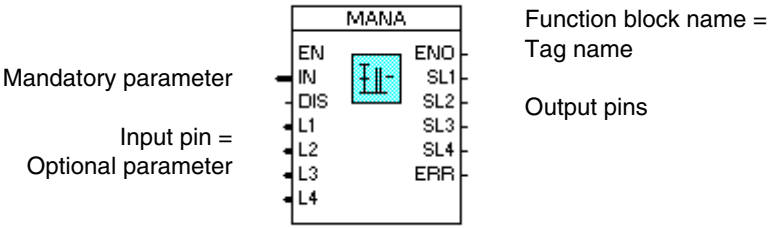
Graphic area — The function blocks and signal flow lines are programmed in the graphic area of the LD program.
The graphic area is provided with a raster to enable elements to be positioned in a straightforward manner, and minimum distances between elements to be preserved. The user can place the corners of function blocks and signal flow lines only within this raster. Variables and constants can be placed anywhere in the program, and are displayed and/or edited in a rectangle. Containers for variables and constants have a short and a long version. The raster can be switched on or off.
An LD program can be up to 10x10 pages in size. The separate pages are delimited by dashed lines. Care should be taken to ensure that no objects are positioned on the dashed lines, as they would be split up over two pages in the documentation.

State line — Display of the current program status

F 2.2 Ladder Diagram menu structure

LD program	*Save*	***Edit***	*Undo*
	Documentation		*Change data type...*
	Check		*Parameters...*
	Header		*Change number of inputs*
	Comment		*Zoom to user FB*
	Exit		*Select variable*
			Access via process display
			Cut
			Copy
Blocks	*Analog*		*Paste*
	Binary		*Delete*
	Constants		*Export block...*
	Converter		*Import block...*
	Acquisition		*Draw lines*
	Arithmetic		
	Controller	***System***	*Variable list*
	Standard		*Tag list*
	Open-loop control		*Hardware structure*
	Modbus master		*Structured data types*
	Modbus slave	***Cross-references***	*Cross references*
	Monitoring		*Find next*
	System functions		*Find previous*
	TCP/IP Send and Receive		
	User function blocks	***Options***	*Version*
	User menu		*Hardcopy*
			Raster on
			Define user menu
LD elements	*Contact*		*Colors*
	Coil	***Back!***	
	Variable		
	Jump	***Help***	*Contents*
	Return		*Overview*
	Label		*Use help*
			Information

F 2.3 Displaying the function blocks



Border — The frame of the function block delimits its select field. You can see from its **color** whether it is selected. The color displayed can be altered, as described on **Page F-11, Changing colors in the program**.

Function block name — Unlike the functions, all function blocks are displayed with a **tag name** (max. 16 characters). All the block names are also included in the system-wide **tag list**. The font color used for the function block name is used for identifying its edit state (**enable/disable**), and can likewise be altered.

Icon — The block type is symbolized by an icon in the case of function blocks, and by a function code in the case of functions.

Input/output pins — A distinction must be made here between inputs and outputs. In accordance with the signal flow, inputs are always displayed on the left and outputs on the right. As with the signal flow lines, the **color** and **line width** conveys information about the data type required/specified.

Mandatory/optional parameters — Unlike the optional parameters, the mandatory parameters require supply via a signal flow line in order to allow the function block to work correctly. To distinguish between them the terminals of the optional parameters are represented as shorter. By defining constants as parameters certain optional parameters are no longer needed at all.

Terminal designation — In a function block each input/output pin also has a code that represents the function of the pin, e.g. **EN** for **enable**.

F 2.4 Changing the defaults

F 2.4.1 Switching the raster on and off

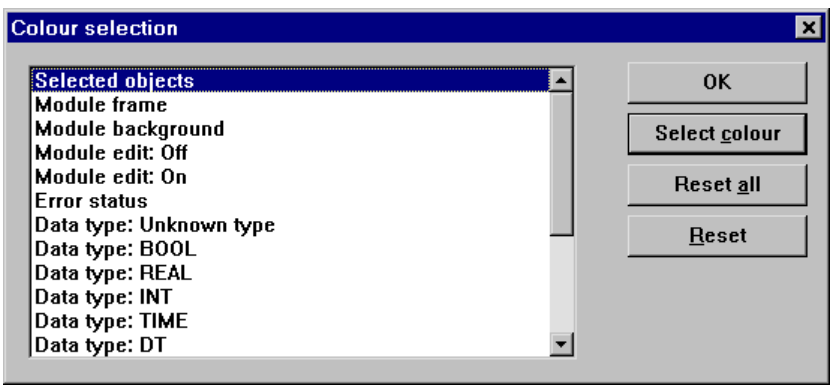
Options → Raster on (→ Save!)

The positioning raster in the graphics area is switched on if previously it was switched off, and vice versa. Any change to this setting remains in force until a different window is opened. If the new setting is intended to be applied more widely than this, then the program should be **saved** after the change to the raster.

The setting that was saved for the last program to be edited is offered as a default. The raster is set to on for the first program to be created in a new project. The spacing of points in the raster cannot be altered.

F 2.4.2 Changing colors in the program

Options → Colors
→Select object whose color is to be changed
(e.g. color of block background)
→Select the required color



tn031us.bmp

SELECT COLOUR	The color can be chosen for the selected object. The default color is selected in the list.
RESET ALL	All object Colors are reset to the defaults.
RESET	The color of the selected object is reset to the default.

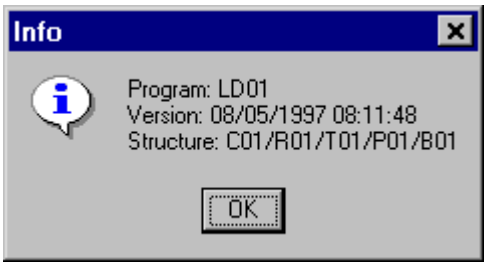
F 2.5 Displaying program information

F 2.5.1 Program version, and assigning it to the project

Options → *Version*

The following details are displayed: the program name, the date the program was last changed (version) and the assignment of the program to the project, resource, task and program list.

The program assignment can be displayed in the form of **long text** or **short text**, as set under Options in the project tree.



tn032us.bmp

F 2.5.2 Program state

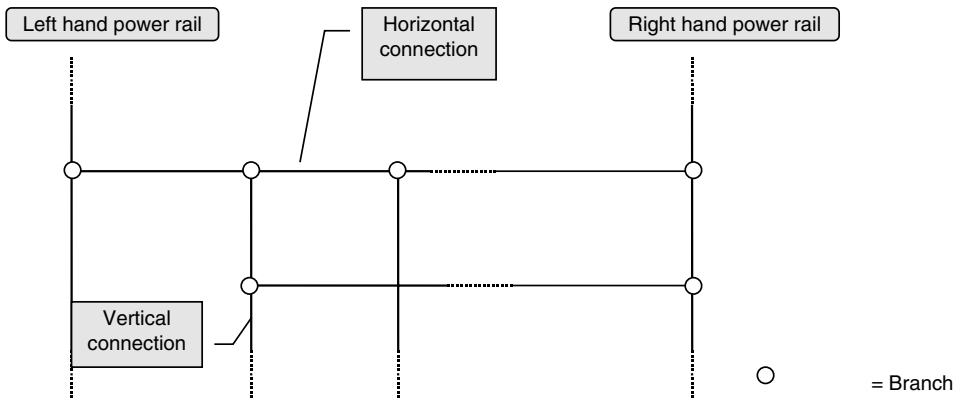
The **state line** shows the name of the program currently being edited, the editor position and the current user.

Editor Position (4,1) Shows side (line, column) currently being edited.

F 3 Description of the Elements of Ladder Diagram

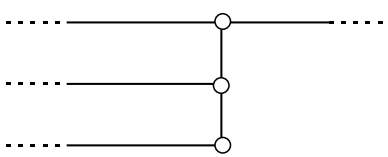
F 3.1 Connections and lines

Horizontal connections can attached to the power rails, and following this, vertical connections can also be made. A connection can have a state of either 0 or 1, which characterizes the current flow. Connections are drawn as horizontal or vertical lines.



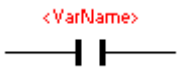
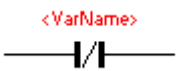
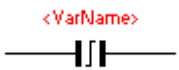
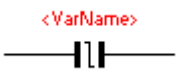
Function	Description
Horizontal connection	Transports the state at the left-hand end to the right-hand end.
Vertical connection	Links all the states on the left-hand horizontal connection with a logical (inclusive) OR and applies the result to the horizontal connections on the right-hand side [1] (wired or).
	[1] As far as the processing sequence is concerned, this means that all the results on the left-hand side must be available.

It should be noted that the following expression is valid in LD (for the current flow), but is not valid in FBD.



F 3.2 Contacts

A contact links the state of the left-hand horizontal connection with the Boolean function of an assigned variable, whereby the value of the assigned variable is not modified. There are two types each of static and transition-sensing contacts.

Symbol	Description/Function
	**Normally-opening contact** The contact is switched when the assigned variable is TRUE
	**Normally-closing contact** The contact is switched when the assigned variable is FALSE.
	**Positive transition-sensing contact** The contact is switched when the assigned variable has a positive transition.
	**Negative transition-sensing contact** The contact is switched when the assigned variable has a negative transition.

The state of the **right-hand** side of a **positive transition-sensing contact** can be obtained from the following table:

		Previous state of the right-hand side <VarName>	
		0	**1**
Current state of the right-hand side <VarName>	0	0	0
	1	State of the left-hand side	0

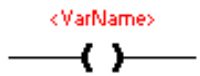
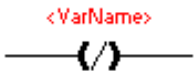
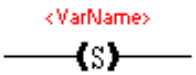
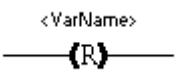
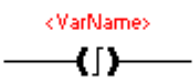
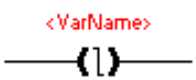
The state of the right-hand side of a **negative transition-sensing contact** can be obtained from the following table:

		Previous state of the right-hand side <VarName>	
		0	**1**
Current state of the right-hand side <VarName>	0	0	State of the left-hand side
	1	0	0

As the user specifies the first value in the variable list for <VarName>, there is no need to define any specific cold-start procedure, which means that both contacts spark or make their state available when an LD program is calculated for the first time by means of appropriate initial values in the assigned variables.

F 3.3 Coils

A coil copies the state of the left-hand connection to the right-hand connection and also stores the result of a Boolean function in the left-hand connection to an assigned Boolean variable. There are six different coils: normal and negated coils, setting and resetting coils and two transition-sensing coils. The coils function as follows:

Symbol	Description/Function
	**Normal coil** Applies the state of the left-hand connection to the assigned Boolean variable and to the right-hand connection.
	**Negated coil** Applies the state of the left-hand connection to the right-hand connection and assigns the negation of the state of the left-hand connection to the assigned Boolean variable.
	**Set coil** The assigned Boolean variable is set to TRUE if the state of the left-hand connection is ON, otherwise the Boolean variable is left unchanged.
	**Reset coil** The assigned Boolean variable is set to FALSE if the state of the left-hand connection is ON, otherwise the Boolean variable is left unchanged.
	**Positive transition-sensing coil** Applies the state of the left-hand connection to the right-hand connection. If the last state of the left-hand connection was OFF and the current state is ON, then the value TRUE is assigned to the assigned Boolean variable.
	**Negative transition-sensing coil** Applies the state of the left-hand connection to the right-hand connection. If the last state of the left-hand connection was OFF and the current state is OFF, then the value TRUE is assigned to the assigned Boolean variable.

The value of the assigned variable of a **positive transition-sensing coil** can be obtained from the following table:

		Previous state of left-hand connection	
		0	**1**
Current state of left-hand connection	0	0	0
	1	1	0

The value of the assigned variable of a **negative transition-sensing coil** can be obtained from the following table:

		Previous state of left-hand connection	
		0	**1**
Current state of left-hand connection	0	0	1
	1	0	0

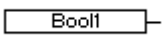
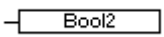
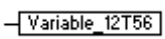
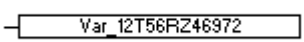
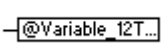
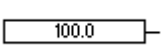
If in both the above cases the previous state of the left-hand connection were assigned cold-start value 0, then only the positive transition-sensing coil could fire in the first calculation cycle. For reasons of symmetry, the initial value of the previous state of the left-hand connection is set to a negative transition-sensing coil 1.

All the contacts and coils come both in a long and a short version. The short version can display at least 10 characters for the assigned variable or constant. In the case of longer labels an overflow indication is represented as '…'. The long version can display the maximum number of characters for a variable or constant. A component of a structured variable can also be declared as the assigned variable.

F 3.4 Variables and constants

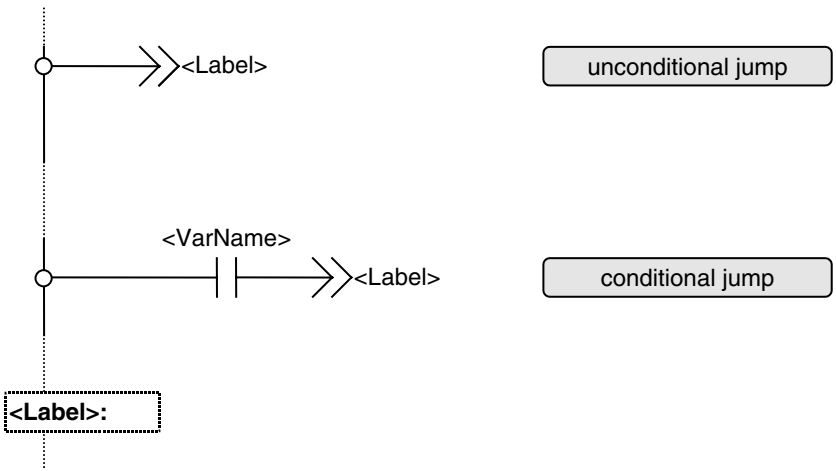
Variables and constants can be placed anywhere in the program, and are displayed and/or edited in a rectangle.

Containers for variables and constants have a short and a long version. The short version can display 10 characters. If the label is longer than 10 characters, the overflow is indicated by '….' The long version can accommodate the maximum possible length of label.
Variables can be read and written either via the process image or directly. Reading or writing via the process image is indicated by @.
Since variables can be placed anywhere in the program, it is essential when inserting them to specify whether they are to be used for reading or writing. Depending on whether a variable or constant is to be used for reading or writing, the surrounding rectangle is provided with either an input or output pin of the appropriate data type.

Symbol	Description/function
	**Variable for reading**
	**Variable for writing**
	**Short version** At least 10 characters can be displayed Overflow indication '…'
	**Long version** Max. possible label length
	Read/write via **process image**
	**REAL Constant**

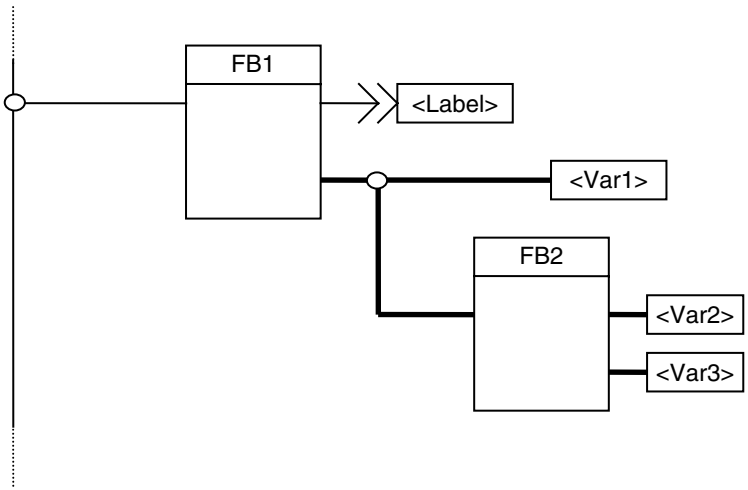
F 3.5 Jumps and returns

One or more jumps and/or returns are allowed in a network. However, these are not executed until the end of network processing.
Where there is more than one jump and/or return, the first one is executed according to the processing sequence.
The targets of jumps are designated by a label. Labels are thus local to a particular program.

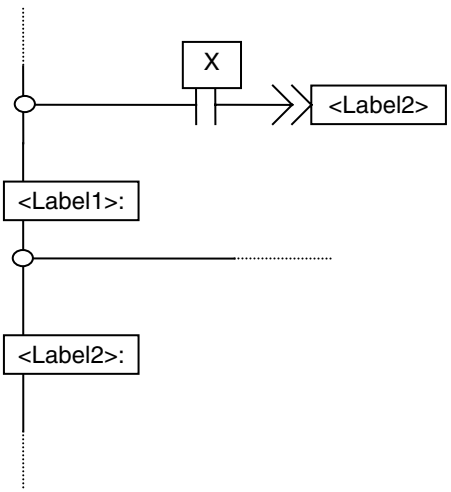


Since jumps are not performed until the execution of the network is complete, a conditional execution must be implemented with several implicit networks.

Example
In the example below all the following actions are performed before the jump: The FB1 outputs are assigned to Var1 and the FB2 inputs, FB2 is calculated, and the FB2 outputs are assigned to Var2 and Var3.



A conditional action before the FB1 outputs are assigned to … could be implemented through the following configuration.

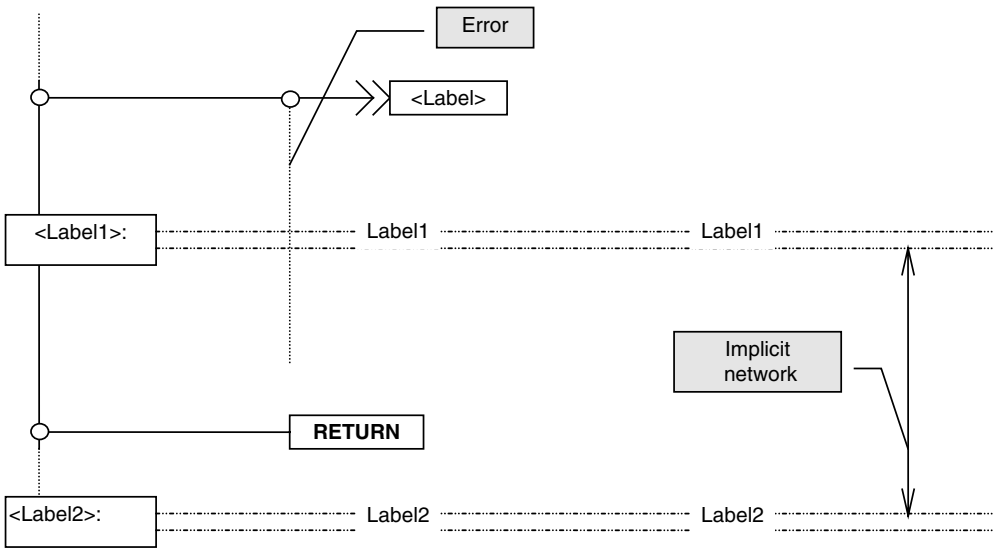


If variable X has the value TRUE, then the network ‘Label1’ is skipped, and processing continues with the execution of network ‘Label2’.

F 3.6 Labels

Labels can be added at any point on the left-hand power rail, and are shown as a double horizontal line with the label name displayed at fixed intervals.

The label can be edited in a rectangle on the left-hand power rail, and can be terminated with a ':'. Connections which pass beyond a network boundary are shown in red and flagged as an error in the plausibility check.



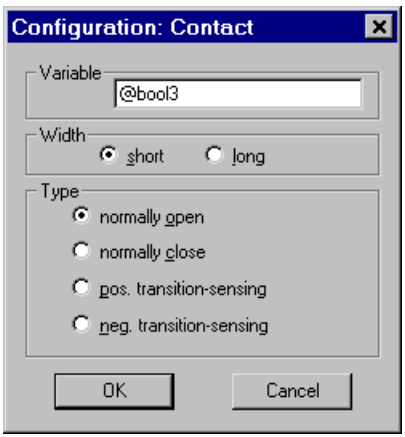
Implicit networks are defined through labels. An implicit network begins at a label or at the beginning of an LD program, and ends at the next label or at the end of the program.

F 4 Defining Parameters for the Ladder Diagram Elements

Parameters are defined for LD elements by selecting the element and then carrying out one of the following actions:

Edit → *Parameters*
→Double-click on the element
→Right mouse-click in the graphic area and select *Parameters* via the Context menu

F 4.1 Defining parameters for a contact



tn024us.bmp

Variable The variable assigned with the contact is configured. By pressing F2, a variable can be selected from the variable list.

Width

short 10 characters can be displayed for the assigned variable. If the variable is longer than 10 characters, this is indicated by '…'.

long The long version, in which variables can be displayed to their maximum length, is chosen for the contact.

Type

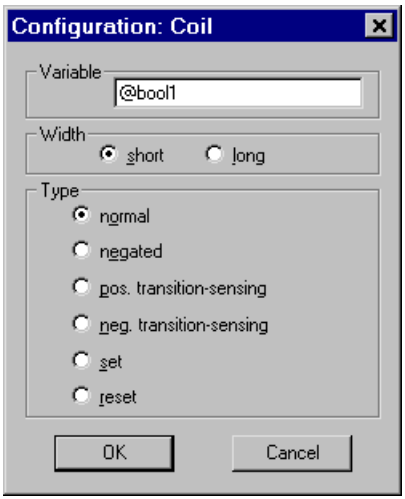
normally open The contact switches when the assigned variable is TRUE.

normally close The contact switches when the assigned variable is FALSE.

pos. transition-sensing The contact switches when the assigned variables have a positive transition.

neg. transition-sensing The contact switches when the assigned variables have a negative transition.

F 4.2 Defining parameters for a coil



tn025us.bmp

Variable The variable assigned with the coil is configured.
By pressing F2, a variable can be selected from the variable list.

Width

short 10 characters can be displayed for the assigned variable. If the variable is longer than 10 characters, this is indicated by '…'.

long The long version, in which variables can be displayed to their maximum length, is chosen for the coil.

Type

normal The assigned variable is given the value currently at the coil input.

negated The assigned variable is given the negated value of the signal at the coil input.

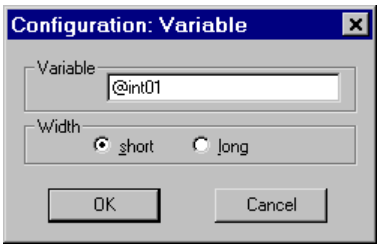
pos. transition-sensing If there is a positive transition at the coil input, then the assigned variable is set to TRUE. Otherwise it is given the value FALSE.

neg. transition-sensing If there is a negative transition at the coil input, then the assigned variable is set to TRUE. Otherwise it is given the value FALSE.

set If the coil input is TRUE, the assigned variable is also set to TRUE. Otherwise the value of the variables is not altered.

reset If the coil input is TRUE, the assigned variable is set to FALSE. Otherwise the value of the variables is not altered.

F 4.3 Defining parameters for variables



tn026us.bmp

Variable The name of the variable is configured.
By pressing F2, a variable can be selected from the variable list.

Width

short 10 characters can be displayed for the assigned variable. If the variable name is longer than 10 characters, this is indicated by '…'.

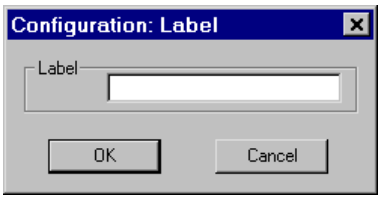
long The long version, in which variables can be displayed to their maximum length, is chosen for the variable.

F 4.4 Defining parameters for a jump

tn027us.bmp

Label The name of the label which forms the target for the jump.

F 4.5 Defining parameters for a label



tn028us.bmp

Label The label name.

F 4.6 Defining parameters for function blocks

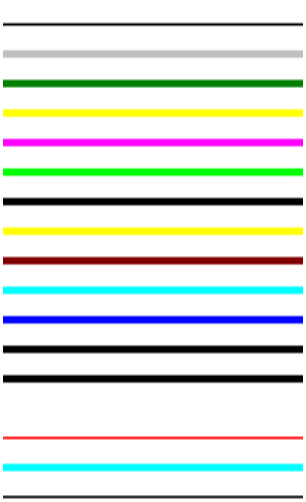
When defining parameters for function blocks, the same procedure is used as in the FBD editor. See also **chapter Function Block Diagram, Parameter definition of function blocks.**

F 5 Editing a Ladder Diagram Program

F 5.1 Representation of the signal flow lines

If the signal flow line is at edit state **selected, incorrect** or **not connected**, then this is displayed, otherwise it shows the transported data type.
The state or transported data type of the signal flow line can be recognized by the width and color of the line, and the color can be set according to the user's preference (see **page F-11, Changing colors in the program**).
The table below shows the connection between data type, edit state, line width and the default color:

Data type/ Processing state	**Color**	**Display**
BOOL	black	narrow
BYTE	gray	wide
DINT	grass-green	wide
DT	dark yellow	wide
DWORD	magenta	wide
INT	light green	wide
REAL	black	wide
TIME	light yellow	wide
UDINT	brown	wide
UINT	turquoise	wide
WORD	dark blue	wide
STRING	black	wide
STRUCT	black	wide
Error state	red	narrow
selected objects	turquoise	
not connected	black	narrow



di0152.bmp

F 5.2 Drawing lines

The LD editor has a special drawing mode in which it is possible to draw horizontal and vertical lines. Drawing mode is activated as follows:

→ *Edit* → *Draw lines*
or
→ Right mouse button (Context menu) → *Draw lines*
The mouse cursor changes into a cross.

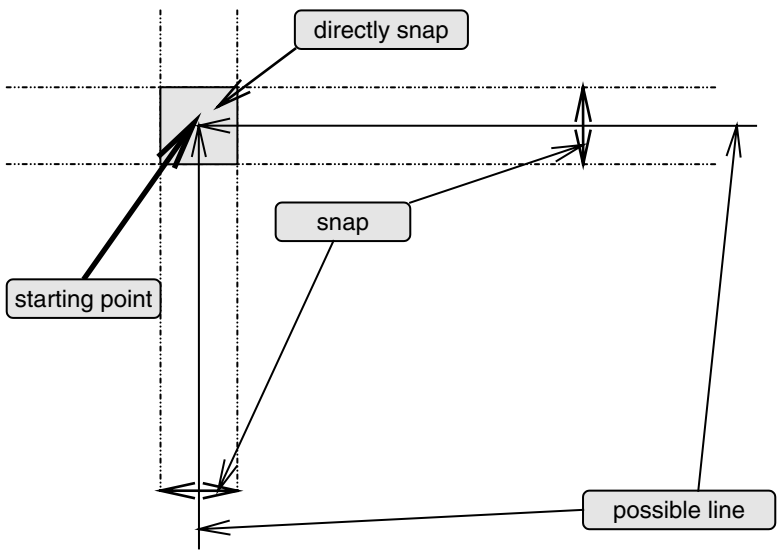
A single mouse-click identifies the beginning of the line, and when the mouse is moved either a horizontal or vertical line is drawn if the cursor is within the snap and as long as the line does not cut across any coil, jump, return, block or network boundary.

Another mouse-click identifies the end of the line and the start of a new line. A mouse-click directly on the snap of the starting point of a line or outside the snap finishes a line.

Dragging a line

Defines start and end of line with a mouse-click
or
by simultaneously pressing CTRL and the left mouse button.

The following diagram clarifies line-draw mode. The snap is exactly 2 raster units in width.

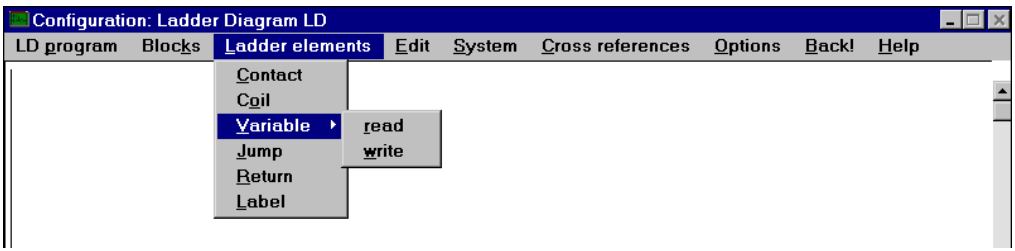


To deactivate draw mode:

Right click or ESC key

F 5.3 Inserting LD elements and function blocks

LD elements → Select elements to be inserted



tn021uk.bmp

After selection an element to insert the cursor is then displayed as an outline of this element, and can be positioned with a left mouse-click. There are no restrictions on where the selected elements may be positioned. Contacts, coils, jumps and returns can be 'dropped' via existing Boolean lines by pressing the left mouse button, whereupon they are fitted into existing lines.
If the element will not fit in, the outline continues to be displayed and a warning tone is emitted.

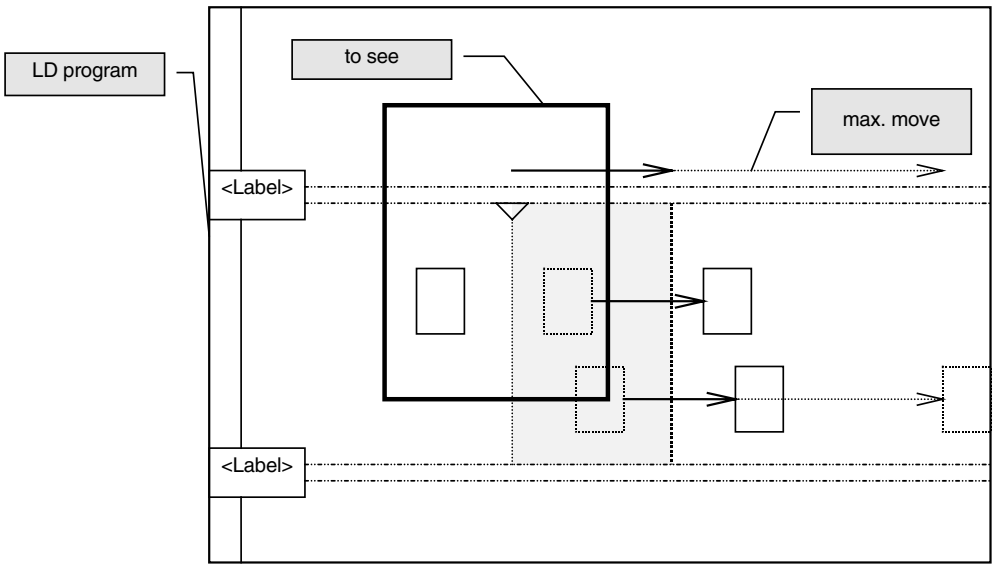
If the placing was performed successfully, the outline cursor is retained, and further elements of the type just selected can be inserted.
The insertion process is brought to an end by clicking the right mouse button.

F 5.4 Inserting columns

Insertion of columns relates only to the current implicit network. Clicking the left mouse button on a network boundary (label) or on the upper or lower border of the display will bring up a triangle pointing either up or down and with a horizontal dashed line.

This triangle can be shifted to the right one raster line at a time. This has the effect of inserting a corresponding number of columns and shifting the partial network to the right of the vertical line by a corresponding number of raster units to the right.

If the mouse moves as far as the edge of the visible part of the display, then the visible region scrolls. The triangle can only be moved if the partial network to be moved is not touching the right-hand edge of the program and if the vertical line do not intersect with a network element other than a horizontal connection. The diagram below should further clarify the procedure for inserting columns.



Horizontal connections are extended accordingly when columns are inserted.

F 5.5 Inserting rows

The insertion of rows corresponds to the insertion of columns. The triangle is positioned at the left or right-hand edge of the visible area. The movement markings run in a horizontal orientation. When rows are inserted, vertical lines are extended accordingly.

F 5.6 Block operations

F 5.6.1 Select block

The mouse is dragged to form a rectangle in the graphic area. All the graphic elements lying wholly within this rectangle are selected.

With the SHIFT key held down, the mouse is clicked on the elements to be selected.

The cursor keys are used to move the cursor until it is over the element to be selected, then, with the SHIFT key held down, the space bar is pressed.

Selected elements are normally displayed in a turquoise color, but the color for selected elements can be changed via the Options menu (see **Page F-11, Changing colors in the program**).

F 5.6.2 Copy

Context menu (right mouse button) → *Copy*
or
Edit → *Copy*

Press key combination CTRL + C

Copy has the effect of transferring the selected elements to an internal storage location. Elements transferred there through a previous Copy are overwritten. Whether or not there are currently any elements in the internal store can be seen from the menu choice *Insert* in the *Edit* or Context menu. If this menu choice is disabled, this indicates that the internal store is empty.

The selected elements can only be copied within the same LD program. It is not possible to copy them into a different LD program, as that would involve quitting the LD editor in order to call up another LD program via the project tree.

If a block is required for use in another LD program, it must be exported and then re-imported into the target program.
When function blocks are copied, the parameter data remain unchanged. However, the tag name is deleted for the copy, as it must be unique.

F 5.6.3 Cut / Delete

Context menu → *Cut* or *Delete*
or
Edit → *Cut* or *Delete*

Press SHIFT + DEL to cut, or DEL to delete

After the selected elements have been cut, they can then be re-inserted in the program using Paste. However, this applies only to copying within a program. Cutting has the effect of over-writing any elements held in the internal store at the time.

If elements are deleted, they cannot subsequently be pasted. Deleted elements can only be restored by quitting the program without saving.

When function blocks are cut, their parameter data and tag name are transferred with them to the internal store, so that next time they are pasted all the appropriate data are available.

F 5.6.4 Paste

Context menu → *Paste*
or
Edit → *Paste*

Press the key combination CTRL + V or SHIFT + INS

After pasting, a surrounding rectangle with a dashed border appears at the position in which the block was previously cut or copied.

F 5.6.5 Move block

Click on a selected element and hold the mouse button down. The rectangle will then appear around the selected block, and the block can now be moved by moving the mouse. When the destination position is reached, the left mouse button is released again. If it is not possible to paste at the destination position, this is signaled by a warning tone, and the surrounding rectangle remains active.

If the cursor is moved into the rectangle that appears after a block is pasted, it changes into a cross with one arrow for each direction of movement in a horizontal plane, and one for each vertical direction. The block can then be moved, by holding the left mouse button down, moving the mouse to the destination position and then releasing the button. If it is not possible to paste at the destination position, this is signaled by a warning tone, and the surrounding rectangle remains active.

The cursor is moved over a selected element or into the rectangle that is displayed after a block is pasted, and then the space bar is pressed. This changes the cursor into a cross with one arrow for each direction of movement in a horizontal plane, and one for each vertical direction. The block can then be moved using the cursor keys or mouse. The block is pasted at the destination position by pressing the space bar. If it is not possible to paste at the destination position, this is signaled by a warning tone, and the surrounding rectangle remains active.

F 5.6.6 Import block

Context menu → *Edit* → *Import block*
or
Edit → *Import block*

A “File open” dialog box appears, containing a list of all the files that have been generated through Export Block with the LD editor. Once a file has been selected, the block is imported, and the rectangle surrounding the block appears. This must then be moved to a suitable position.

Imported variables that are not yet included in the variable list are displayed in red.

F 5.6.7 Export block

Context menu → *Edit* → *Export block*
or
Edit → *Export block*

A “File save” dialog box appears, containing a list of all previously-exported files in the most recently selected export directory.
Tag names are not exported.

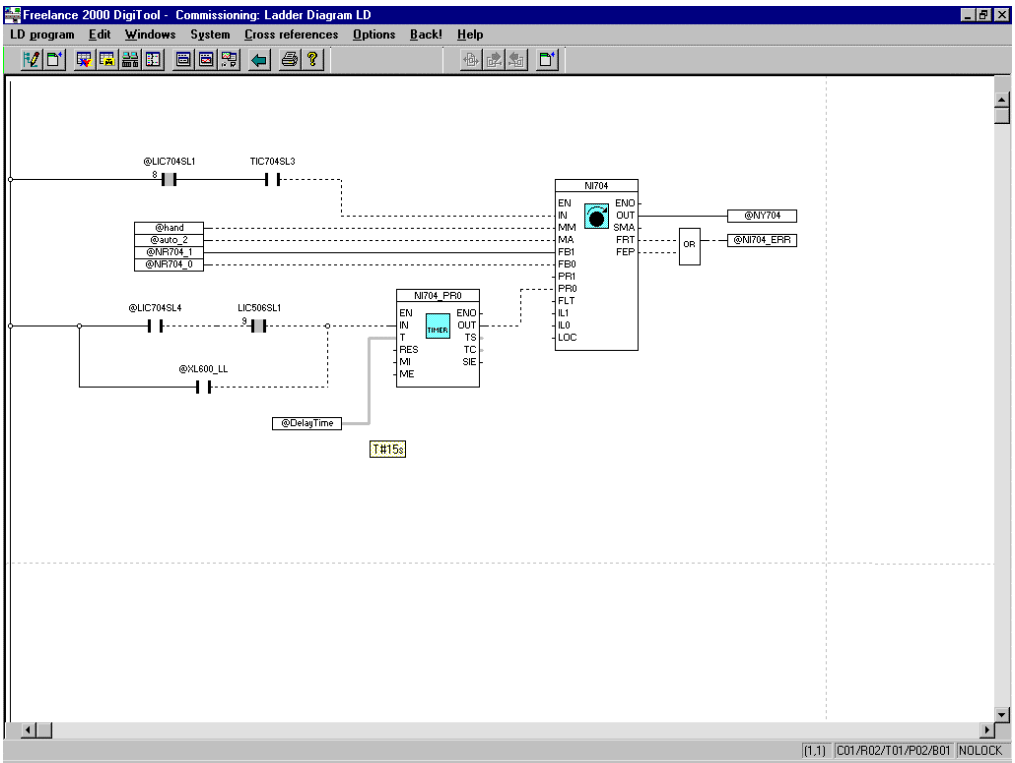
F 5.6.8 Undoing an action

Edit → *Undo*
or
Context menu → *Undo*

This function enables the last action performed to be undone. Irrespective of this function, the program state remains **incorrect** until the next plausibility check.

F 6 Commisioning the Ladder diagram (LD)

On commissioning the ladder diagram, the program is displayed in the same way as in configuration except that in commissioning mode the program cannot be modified structurally.



tn035us.bmp

Individual function blocks can be selected and parameters set for them. Operating modes can also be called up and modified from commissioning mode
Thereafter, certain program test functions are available to whoever is commissioning the system.

Boolean values (binary values) are initially displayed directly with their logical state of 1 or 0.

logical 1 ——————————— true
logical 0 ------------------------------ false

When the variables or terminals of a block are overrun, the current calculated values should be read.

After this, values within a cycle can be defined only once. Function block pins can also be defined to analog or binary values.

Input pins of function blocks which are not loaded can thus be assigned permanent values. This can be difficult notice later and should therefore be used with caution.

Click right mouse button on variable or function block pin → Input values → OK

The writing of a value should not be confused with forcing in the I/O module. The value written can be overwritten by the program in the next cycle.

F 7 Variable List and Tag List

F 7.1 Entries in the variable list

System → Variable list

This has the effect of switching to the variable list. For details, see **chapter Variables**.
The variable list contains all the inputs, outputs and flags used in the system. A variable can be selected in the list, copied and than insert into the program.

F 7.2 Entries in the tag list

System → Tag list

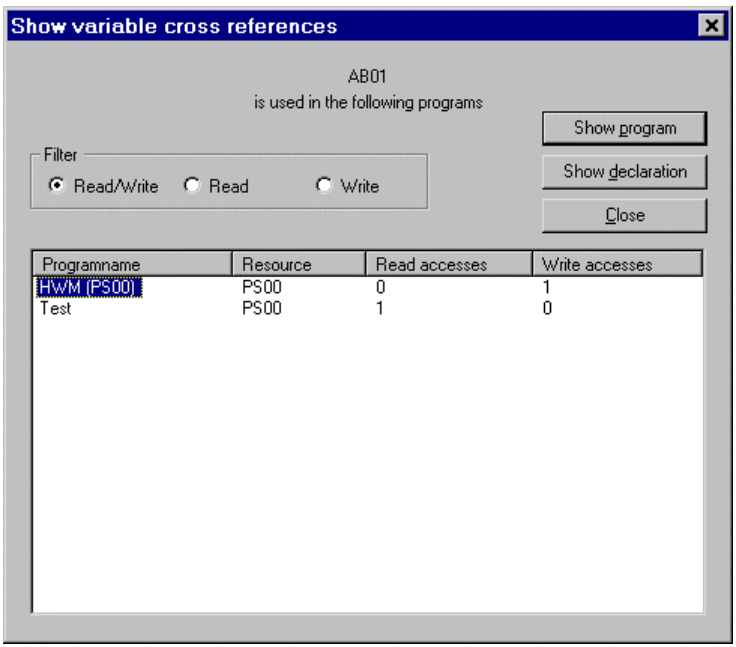
The tag list is called up. For details, see **chapter Tags.**
The tag list contains all the tag names allocated in the system.

If tag names assigned to function-block calls in the LD program have been deleted in the tag list, then the entries in the corresponding Parameter definition and configuration screens will be blank on returning from the tag list, and they will need to be re-entered.

F 8 Cross References

Cross references can be selected directly from inside the FBD program.

- Select a variable, I/O component or tag
- Menu item *Cross reference* or the F5 function key.



tn024uk.bmp

This window has a number of sorting and filtering options; the window settings are saved. In contrast to the variables, for the tags there are no read and/or write accesses defined.

SHOW PROGRAM — Display a program with preselection of these variables, or display a module with preselection of the I/O components.

SHOW DECLARATION — If a variable or I/O component has been assigned, then switch into the I/O editor and to the corresponding I/O components, or to the variables in the variable list.

FILTER — A filter makes possible display of only those varibles that are to be read or written by the programs in question.

After activation it is possible to go to the programs that have been listed as cross references. If that is done, and then in that program the menu choice *Back!* is chosen, a return jump is made to the original editor.

Show next/previous cross reference

- Select a variable, menu: *Cross references* → *Find next* or *Find previous*

The next or the previous point of use of the selected variable in the current program is shown.

F 9 General Processing Functions

F 9.1 Saving the program

LD program → Save

The program is saved without exiting. Even if a program is incorrect it can still be saved, and then be corrected at any time in the future.

If the program is not saved in the project tree either when it is closed or beforehand, any change(s) made to the program will be lost.

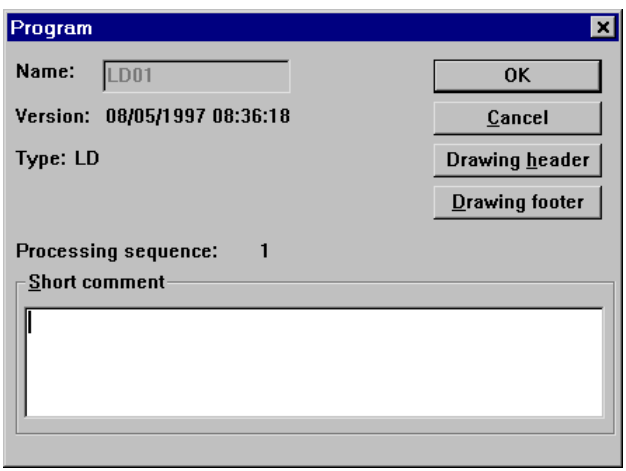
F 9.2 Documenting the program

LD program → Document

This has the effect of switching from the program to the documentation manager. Here the project documentation is defined and output for a specific user. For details see **Engineering Manual, System Configuration, Documentation**.

F 9.3 Program header

LD program → Header



tn030us.bmp

A brief, program-specific comment for the header line of the program documentation can be entered or edited.

Drawing header / footer — See **Engineering Manual, System Configuration, Documentation**

F 9.4 Edit program comment

LD program → Comment

A longer program-specific comment for describing functionality can be edited here.
See **Engineering Manual, System Configuration, Project Manager, Creating Comments**.

F 9.5 Back!

Back!

Exits the LD program and calls the application from which the switch to the LD program was performed (single-step return).

F 9.6 Exit Ladder Diagram

LD program → Exit

Exits the LD program and calls the project tree.

F 9.7 Produce hardcopy

Options → Hardcopy
→ Confirm printer setting in the Windows print mask (Setup).

The contents of the screen are dumped to the printer.

F 9.8 Program elements plausibility check

Select program element
→LD program → Check

AI entries relevant to functioning are checked for syntactical and contextual correctness. Any errors or warnings are displayed in the form of an errors list. If any errors are found by the plausibility check, then the processing state of the program element is **incorrect**.

Newly-entered, copied or moved program elements have the processing state **incorrect**.

F 9.9 Deleting an LD program

Project tree → Select program → *Edit* → *Delete*

The variables and tag names are retained in other programs and in the variables/tag list, and can be assigned again.

F 9.10 Copying and pasting an LD program

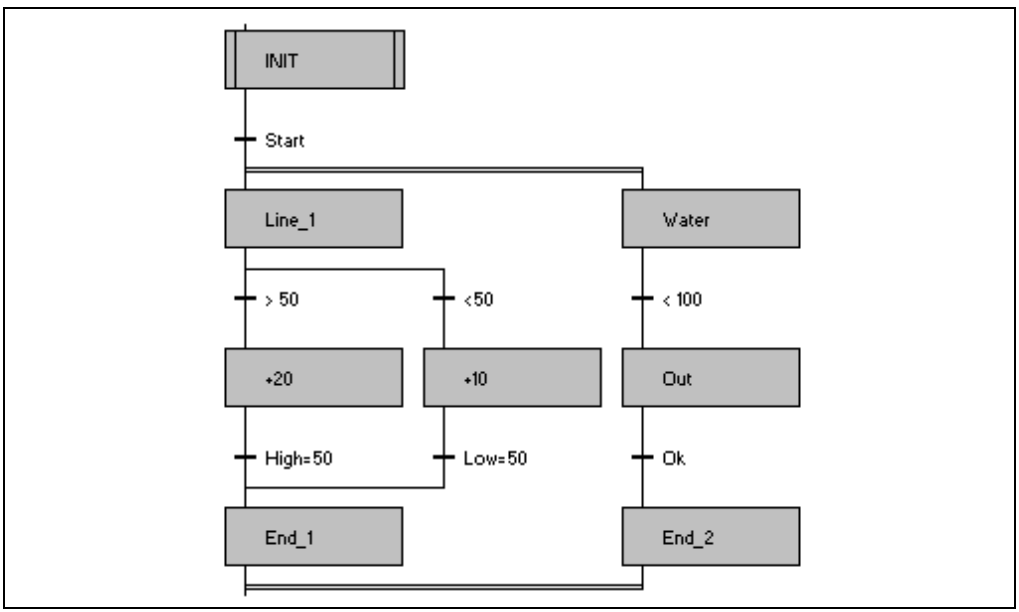
Project tree → Select program to be copied → *Edit* → *Copy* or CTRL + C
→Select position to which the program is to be copied
→*Edit* → *Paste* or CTRL + V
→Select below, above or level depending on the position selected
→Allocate program name

The program is copied and assigned to one of the project's program lists under a **new, unique name**. The program is copied in its configured form, including program header and program comment. The tag names of the function blocks **are not copied** along with the program. The copying of an LD program does not affect the declaration of variables or tag names. The copied program is identified as **incorrect**. The date and time the program was copied form its version identifier.

F 9.11 Linking programs

Through variables, programs are linked either with one another or with the input / output cards

G Sequential Function Chart (SFC)



Contents

G 1 General Description - Sequential Function Chart (SFC)

The sequential function chart is an IEC 61131-3 programming language for creation and modification of sequence controls. The sequential function chart enables one to structure and display complex tasks in a clearly arranged manner. The structure is similar to a network of elements, with the individual elements of the sequential function chart denoting the subtasks of the user program.

The subtasks are described in the programs which are assigned to the transitions and steps. These programs can be generated in the function block diagram (FBD), ladder diagram (LD) or instruction list (IL). A transition describes the step-enabling condition for activation of the next step. The steps are then processed cyclically until the next transition is fulfilled.

The transitions are linked via lines and branches, controlling processing of the individual elements. A distinction is made between alternative and parallel lines or branches. In the case of sequence selections, only one string is processed in each case, whereas several steps are processed concurrently in the case of simultaneous sequence divergences.

Since a step is processed only until the following transition is enabled, advantages are procured as regards the CPU engagement because only very few steps can be active simultaneously. However, functions or function blocks, which must be computed continuously e.g. analog monitoring for alarm value messages, cannot be configured directly in the SFC programs, since they can no longer be computed when the SFC switches forward. These programs are entered into the program lists and are processed cyclically.

The SFC program can be activated automatically as a function of an enable and start-time specification. A new start time and repeat time permit selective repeats of the entire SFC program.

The SFC program is processed in the Manual of Automatic modes, while there are also possibilities for regulating the course of the SFC program via operator interventions. All operator interventions can be individually interlocked. Using the supplementary package DigiLock, operation of the entire SFC can be assigned to individual user groups or interlocked for certain user groups.

See also **Engineering Manual, System Configuration, Commissioning**, also **Operator's Manual, Operator Station, Sequential Function Chart** and manual **DigiLock.**

Configuration of the SFC program permits easy positioning and linkage of steps and transitions. It is syntax-oriented, i.e. elementary parts of the sequential function chart such as identifier can only be stated correctly. To support programming, the editor is divided into lines and columns in which in each case only certain elements of the sequence flow chart can be programmed.

The operating range consists of thin and thick lines. The thin lines are used only for making horizontal or simultaneous sequence selections. The broad columns are destined for steps and

transitions. The maximum number of lines per program and the maximum number of columns are limited to 512 and 16, respectively. All function-relevant entries can be checked for plausibility on request.
Displays can be assigned to each step or transitions via their own assignment editor, and these displays can be then called from DigiVis via the Display Selection dialog. In this manner, the user can get a better orientation and call up the displays relevant to the process.

Criteria windows can also be defined for the operator interface, in order to give the user instructions for the current processing activity. Any arbitrary variables can be depicted in these windows, featuring their current values and a comment. A tag can also be assigned to each variable. This enables a relevant faceplate to be called up on the operator station in the operation dialog of the criteria window.

☞ During configuration it should be borne in mind that the sequential control display in the operation and observation mode can only be illustrated with 6 columns.

☞ Since in SFC only 5 programs (FBD, LD and/or IL) per step can be incorporated, edit steps must be configured as one step in order to safeguard a clear sequence control.

G 1.1 Creating a new SFC program

SFC programs are created in the project tree. For detailed description see **Engineering Manual System Configuration, Project tree.**

Select project tree → *Edit* → select a task → *Insert next level* → *SFC program*

Select project tree → *Edit* → select a program list → *Insert above/below* → *SFC program*

G 1.2 Calling up a SFC program

An SFC program can be called up from the following program parts:

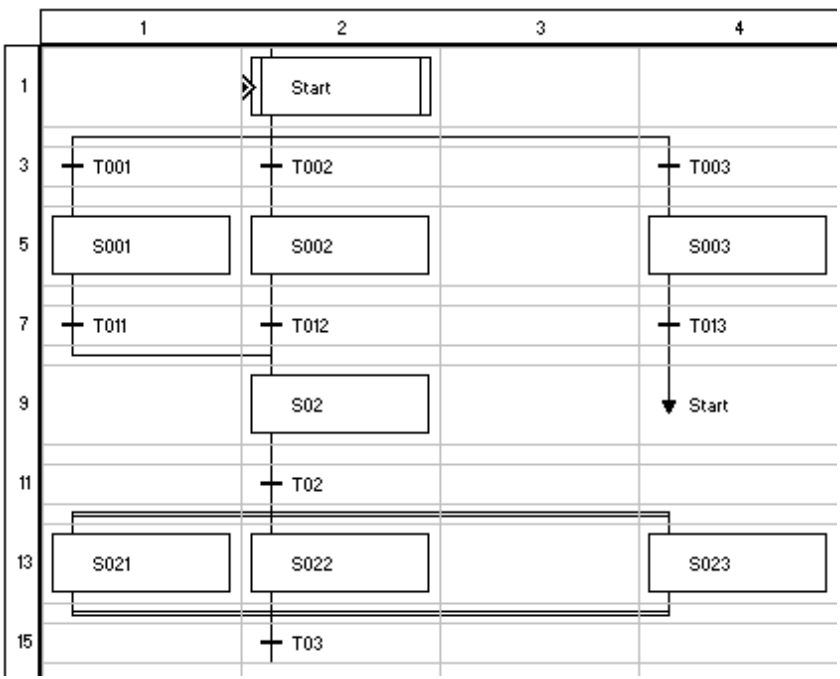
Select project tree → select SFC program → *Edit* → *Program*

Double click on program

G 1.3 Basic rules

- A sequence control always begins with **one** initial step
- A step always follows a transition and vice-versa.
- **Only one** transition is possible before and after a simultaneous sequence divergence.
- After start and end of a sequence selection convergence, several transitions always follow.
- A branch is always closed in the same manner as it was opened.
- The last element of an SFC program must always be a transition.

G 1.4 Example of how to edit



di1401.bmp

The following example is given to help better explain the structure of a sequence control. In the explanation reference made always refer to the lines and columns in the example.

A sequence control always begins with an *initial step* (→ **line 1, column 2**).

After that and just as in every step, a sequence selection divergence may follow. Under the step, a *sequence selection divergence start* is placed (→ **line 2, column 2**), for every other alternative sequence selection divergence, a *sequence selection divergence add* is placed (→ **line 2, columns 1+4**).

To bridge columns it is possible to set *horizontal seq. selection line* **(→ line 2, column 3)**.

After every branching follow *transitions* (→ **line 3, columns 1+2+4**).

Since steps can only be entered in the thick lines, the next thin line is bridged with a *vertical line* (→ **line 4, columns 1+2+4**) to enable steps to be inserted (→ **line 5, columns 1+2+4**).

A union of the sequence selection steps is effected with *sequence selection convergence add* (→ **line 8, column 1**), before or after the next step with *sequence selection convergence end* **(→ line 8, column 2)**.

Goto (→ **line 8, column 4**) can be inserted in the next thick field after a transition.

A sequence selection divergence follows directly after a transition (→ **line 11, column 2**). To begin the divergence of the transition, the *simultaneous sequence divergence start* is selected (→ **line 12, column 2**).

The other simultaneous steps are begun with *simultaneous sequence divergence add* (→ **line 12, column 2+4**).

To bridge columns the function *horizontal simultaneous sequence line* (→ **line 12, column 3**) must be selected.
After inserting the steps or before the next transition the divergence with *simultaneous sequence convergence add* (→ **line 14, column 1+4**) is ended with *simultaneous sequence convergence end* (→ **line 14, column 2**).

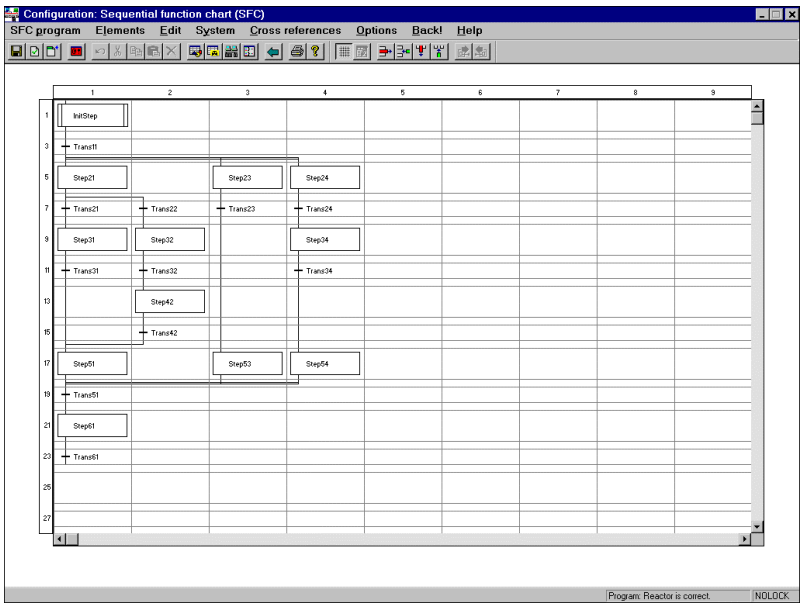
G 2 Structure of the Sequential Function Chart

G 2.1 SFC program user interface

Menu line

Graphics area

Status line



di1402uk.bmp

Graphics area — The items for generating a sequence control are entered in the graphics area. To obtain a better overview, the graphics area is divided up into grids. Grids and scales can be windowed in and out (lines 1-512 / column 1-16).

Status line — Display of program name and state correct/incorrect.

G 2.2 Menu structure

SFC program
Save
Documentation
Check
Header
Comment
Exit

Elements

Initial step	CTRL + I
Step	CTRL + S
Jump	CTRL + G
Transition	CTRL + T
Vertical line	CTRL + E
Horizontal seq. selection line	CTRL + NUM 4
Seq. selection divergence start	ALT + NUM 7
Seq. selection divergence add	CTRL + NUM 7
Seq. selection convergence add	CTRL + NUM 1
Seq. selection convergence end	ALT + NUM 1
Horizontal simultaneous seq. line	CTRL + NUM 6
Simultaneous seq. divergence start	ALT + NUM 9
Simultaneous seq. divergence add	CTRL + NUM 9
Simultaneous seq. convergence end	ALT + NUM 3
Simultaneous seq. Convergence add	CTRL + NUM 3

Edit
Undo
Column insert, Column delete
Row insert, Row delete
Delete element
Parameters of element
Edit program
Define criteria window
Define display selection
Parameters of SFC
Cut, Copy, Paste, Delete
Export block, Import block

Cross references *Find next/find previous*

Options *Version*
Hardcopy
Grid, Scale

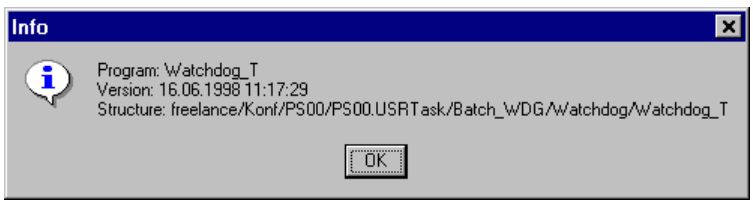
Back!

Help *Contents, Overview*
Use help
About

System
Variable list
Tag list
Hardware structure
Structured data types

G 2.3 Program information

→ *Options* → *Version*



di1430uk.bmp

Insert a window with information on the current SFC program.
Program name, version and the assignment in the project tree are displayed.

G 2.4 Drawing help

Grid

→ *Options* → *Grid*

Grid is the term used to window in and out the spacing of lines and columns. A hook stands in front of the menu item if the grid is shown.

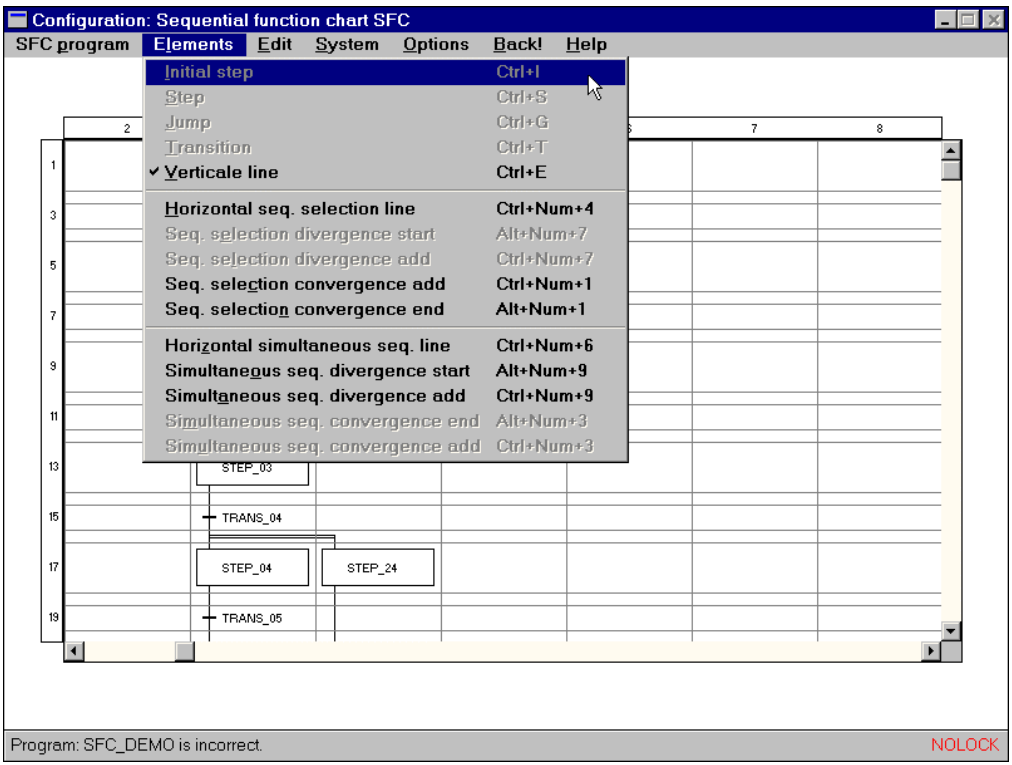
Scale

→ *Options* → *Scale*

Scale is a term used to window in and out the number of lines and columns. A hook stands in front of the menu item if the scale is shown.

G 3 Editing SFC Elements

→ *Elements*



di1406uk.bmp

Since call-up of graphic elements is possible via either the menu, mouse, keyboard as well as the hotkeys, the following points are valid for all possibilities.

Initial step, Step, Jump and *Transition* can only be entered into the thick lines.

The hook in front of an element represents the default value and shows which elements can be entered by pressing the SPACE key. The default value is stored line by line, so that in a line the last element entered can be recalled with the SPACE key.

The given names (max. 8 characters) must be definite throughout the SFC program.

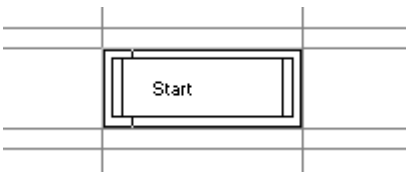
G 3.1 Initial step

Each sequential function chart program begins with an initial step. This is the step of the program which must be accessed on starting the program or with the Return operator intervention.

→ Elements → Initial step

→ CTRL + I

Only one initial step is always permitted in an SFC program.



di1431.bmp

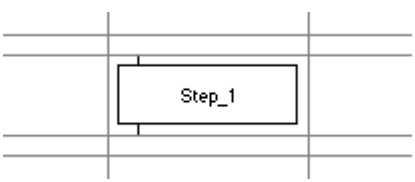
G 3.2 Step

The step describes what is to be controlled in this process step. The actions of the step are described in the assigned programs. These programs can be written in either the instruction list (IL), ladder diagram (LD) or function block diagram (FBD).

→ *Elements → Step*

→ CTRL + S

Insert a step. The name of the step (max. 8 characters) must be unequivocal within the SFC program. Up to 8 programs can be assigned to one step.



di1432.bmp

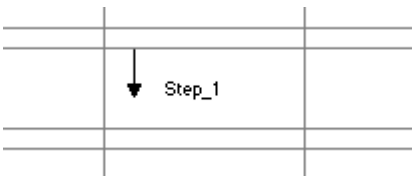
G 3.3 Jump

Jump is one means of going from one branch to another step situated inside or outside this branch. Jump is used instead of a step and is executed by the transition at the jump site. This transition must be fulfilled.

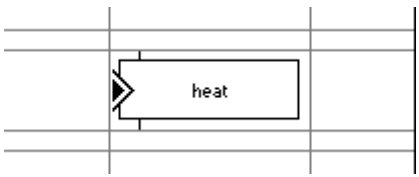
→ *Elements* → *Jump*

→ CTRL + G

How to insert a jump. Insert an arrow on the left of the step of the jump destination, if already available in the SFC program.



di1433.bmp



di1467.bmp

When entering the name of the jump destination, pay attention to capital and small letters!

A branch must always be closed in the manner as it was opened. The SFC structure is incorrect if no element *Seq. selection convergence end* was set on jumping from a branch.

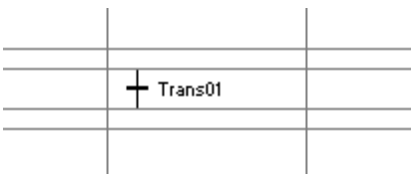
G 3.4 Transition

The transition describes what must be fulfilled in this process step, so that the next step can be activated. The actions of the step are described in an assigned program. This program can be written in either the instruction list (IL), ladder diagram or function block diagram (FBD). The result of the transition (.RESULT) must be logic 1, so that the following step(s) can be activated.

→ *Elements* → *Transition*

→ CTRL + T

How to insert a step-enabling condition. The name of the transition (max. 8 characters) must be stated definitely.



di1434.bmp

G 3.5 Vertical line

Elements can be linked with a vertical line so that a step can be omitted, thus conferring more clarity on a string of the SFC structure.

→ *Elements* → *Vertical line*

→ CTRL + E

Insert a vertical line to make a line of steps and transitions complete.

di1435.bmp

☞ A transition is always followed by a step irrespective of the branch, and vice-versa, a step is always followed by a transition.

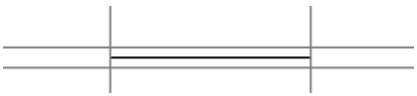
G 3.6 Horizontal sequence selection line

To omit a step and thus confer greater overall clarity on the SFC structure, it is possible to link the elements of a sequence selection with the horizontal sequence selection line.

→ *Elements* → *Horizontal seq. selection line*

→ CTRL + NUM 4

Insert a horizontal sequence selection line to connect a column to the next but one column in a sequence divergence. Can only be inserted in the thin lines.



di1436.bmp

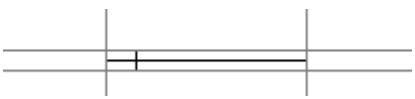
G 3.7 Sequence selection divergence start

In the case of a sequence selection, only one of several strings is computed. Each alternative string begins with a transition. These transitions decide whether or which string is alternatively computed. The start of a sequence selection is always placed after a step.

→ *Elements* → *Seq. selection divergence start*

→ ALT + NUM 7

Open the sequence selection divergence of the preceding step. Can only be inserted in the thin lines.



di1439.bmp

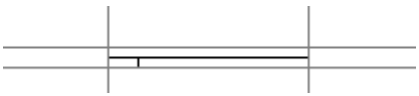
G 3.8 Sequence selection divergence add

Apart from the start of a sequence selection, there is the element *Sequence selection divergence add* with which the number of alternative strings can be increased.

 → *Elements* → *Seq. selection divergence add*

 → CTRL + NUM 7

Open the sequence selection divergence add for the underlying transition. Can only be inserted in the thin lines.



di1438.bmp

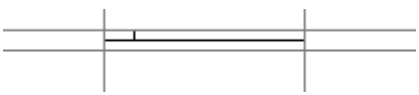
G 3.9 Sequence selection convergence add

Apart from the end of a sequence selection, there is the element Simultaneous sequence divergence start with which the alternative string is closed.

 → *Elements* → *Seq. selection convergence add*

 → CTRL + NUM 1

Close the sequence selection convergence of the previous transition. Can only be inserted in the thin lines.



di1442.bmp

G 3.10 Sequence selection convergence end

In the case of a sequence selection only one of several strings is computed. Each alternative string begins with a transition and ends with a transition. The last transition of the active string decides whether or when the sequence selection is closed. The end of a sequence selection is always placed before a step.

→ *Elements → Seq. selection convergence end*

→ ALT + NUM 1

Close sequence selection convergence end of the previous transition and continue in the sequence control to the next step. Can only be inserted in the thin lines.

di1443.bmp

G 3.11 Horizontal simultaneous sequence line

To omit a string and thus confer greater overall clarity on the SFC structure, it is possible to connect the elements of a simultaneous sequence divergence with the sequence selection divergence start.

→ *Elements → Horizontal simultaneous seq. line*

→ CTRL + NUM 6

Insert a simultaneous sequence line from a column to the next but one column in a sequence divergence. Can only be inserted in the thin lines.

di1437.bmp

G 3.12 Simultaneous sequence divergence start

In the case of a simultaneous sequence divergence all parallel strings of several strings are computed. Only one transition decides whether or when the parallel strings begin. The start of a simultaneous sequence divergence is always placed after a transition.


→ *Elements* → *Simultaneous seq. divergence start*


→ ALT + NUM 9

Open simultaneous sequence divergence start of previous transition. Can only be inserted in the thin lines.

di1445.bmp

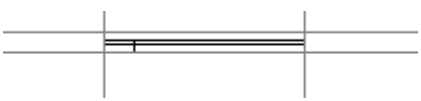
G 3.13 Simultaneous sequence divergence add

Apart from the end of a simultaneous sequence divergence, there is the element Sequence selection convergence add with which the parallel string is closed.


→ *Elements* → *Simultaneous seq. divergence add*


→ CTRL + NUM 9

Open simultaneous sequence divergence add for the underlying step. Can only be inserted in the thin lines.



di1444.bmp

G 3.14 Simultaneous sequence convergence end

In the case of a simultaneous sequence divergence, all strings are computed simultaneously. A simultaneous string always ends with only one transition, with this last transition being placed after the simultaneous sequence convergence, and deciding whether or when the simultaneous sequence divergence is closed. The end of a simultaneous sequence divergence is thus placed before a step.


→ *Elements → Simultaneous seq. convergence end*


→ ALT + NUM 3

Close simultaneous sequence selection after one step. Can only be inserted in the thin lines.

di1441.bmp

G 3.15 Simultaneous sequence convergence add

Apart from the end of a simultaneous sequence divergence, there is the element Horizontal simultaneous sequence line with which the parallel string is closed.


→ *Elements → Simultaneous seq. convergence add*

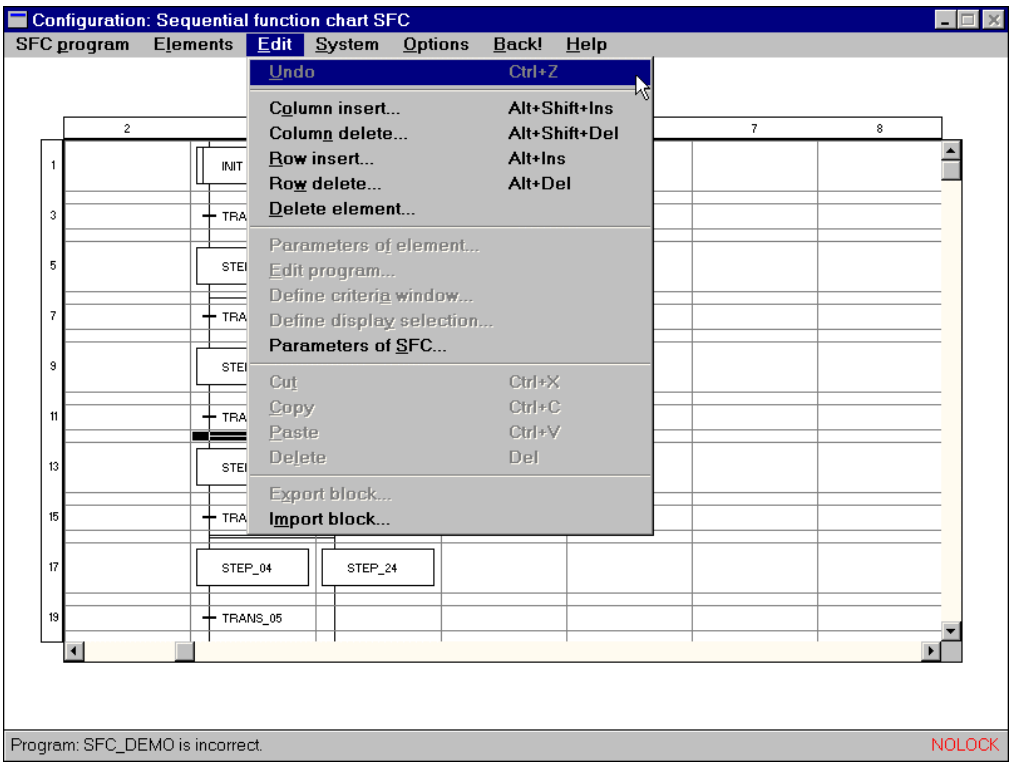

→ CTRL + NUM 3

Close simultaneous sequence convergence after one step and continue in the sequence control to the next transition. Can only be inserted in the thin lines.

di1440.bmp

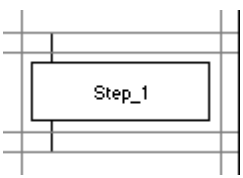
G 4 Edit SFC Program

→ *Edit*



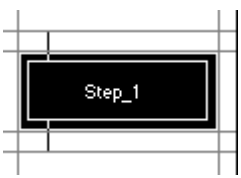
di1407uk.bmp

The following steps require prior selection of an element or a block. A made selection can be recognized by a color inversion of the element or field. **An element is selected** by clicking with the mouse.



di1446.bmp

Element not selected



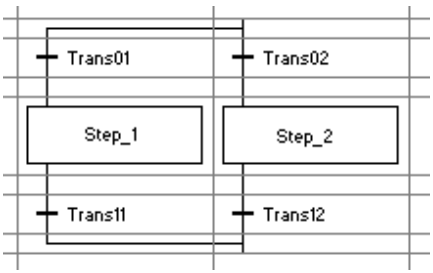
di1447.bmp

Element selected

The following are possibilities for **selecting a block** :

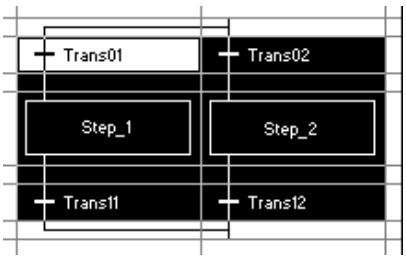
1. Click the first element with the mouse. With a mouse key pressed only a frame can be drawn to contain all elements to be selected. If the frame is drawn only within the first element, this is marked as a block (edges of the field will be black).

2. Click the first element with the mouse and click the next with the shift key is pressed. Everything within the frames of these two elements will be selected.



di1448.bmp

Block not selected

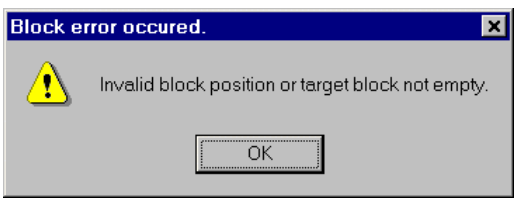


di1449.bmp

Block selected

G 4.1 Shift blocks

After selecting a block, shift it by clicking and keeping the mouse key pressed. The new insertion point can be selected by shifting the marked border lines. If the block cannot be positioned because it would otherwise cover existing elements, an error message will follow and the block can be positioned at another place.



di1450uk.bmp

G 4.2 Undo

→ *Edit* → *Undo*

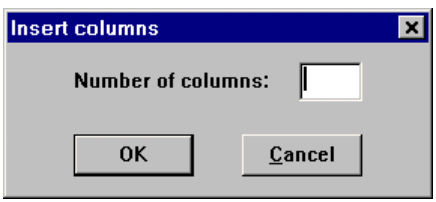
Only the last edited block function, i.e. *cut*, d*elete* or *insert*, can be undone.

G 4.3 Edit columns / lines

Column insert

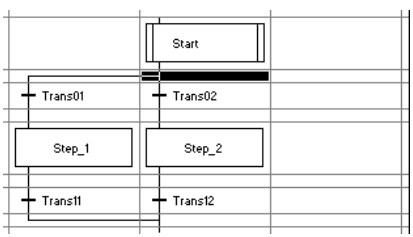
→ *Edit* → *Column insert*

→ ALT + SHIFT + INSERT



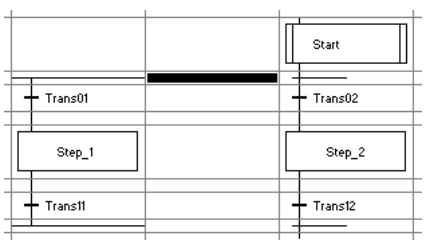
di1408uk.bmp

Every selected menu element can be edited.
An interrogation follows as to how many columns should be inserted. The selected column is shifted to the right by the number of columns to be inserted.
If the number of columns to be inserted is too big, an acoustic signal will follow and the insert will be rejected. A smaller number can now be stated or the whole exercise can be canceled. A message will also be given if due to the insert elements must be shifted down and out of the grid (maximum of 16 columns).



di1451.bmp

After inserting a column



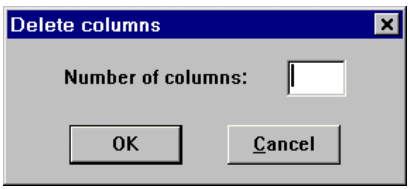
di1452.bmp

After inserting a column

Column delete

→ *Edit* → *Column delete*

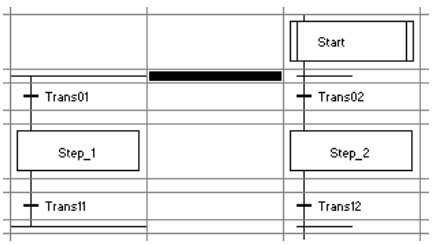
→ ALT + SHIFT + DELETE



di1409uk.bmp

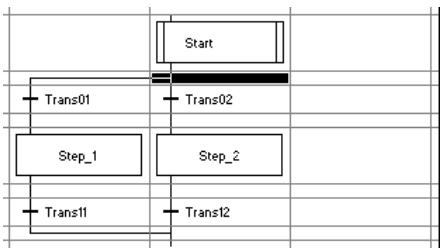
Every selected menu element can be edited.
An interrogation as to how many columns are to be deleted will follow. The number of columns to be deleted, stated on the right side of the elements, are shifted by the same number to the left.

Only columns can be deleted which no longer contain any elements.



di1453.bmp

Before deleting a column



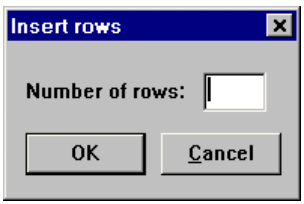
di1454.bmp

After deleting a column

Insert row

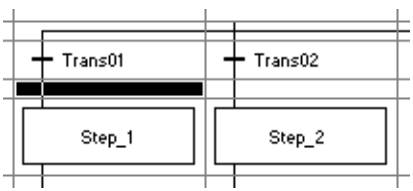
→ *Edit* → *Row insert*

→ ALT + INSERT



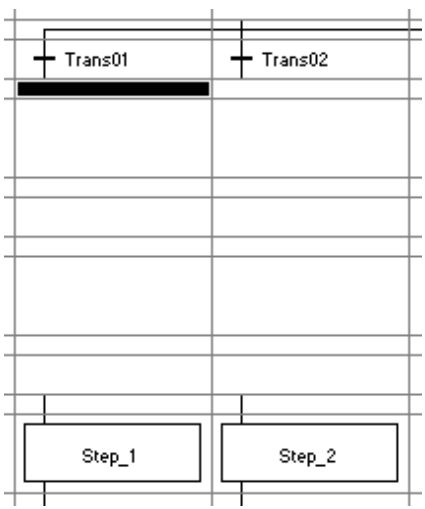
di1410uk.bmp

Every selected menu element can be edited.
An interrogation as to how many rows are to be inserted will follow. The selected row is shifted downwards by the number of rows to be inserted. If the number of rows to be inserted is too big, an acoustic signal will follow and the insert will be rejected. A smaller number can now be stated or the exercise can be canceled. A message will also be given when due to the insert elements must be shifted down and out of the grid (maximum of 512 lines).



di1455.bmp

Before inserting two rows



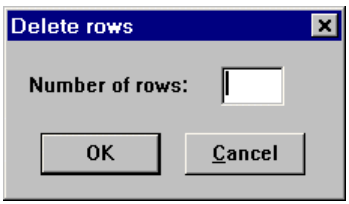
di1456.bmp

After inserting two rows

Delete row

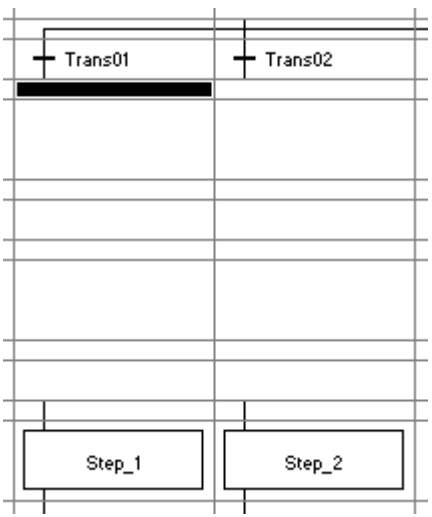
→ *Edit* → *Row delete*

→ ALT + DELETE



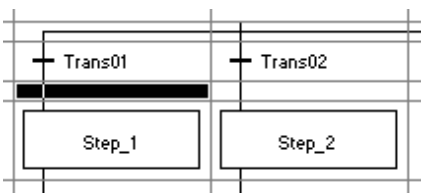
di1411uk.bmp

Every selected menu element can be edited.
An interrogation will follow as to how many columns should be deleted. The elements which are located underneath the number of rows to be deleted will be shifted upwards by the stated number of rows.



di1457.bmp

Before deleting two rows

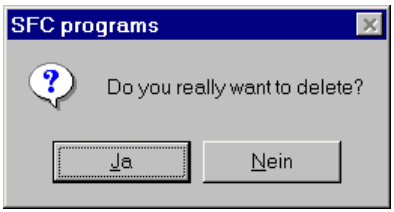


di1458.bmp

After deleting two rows

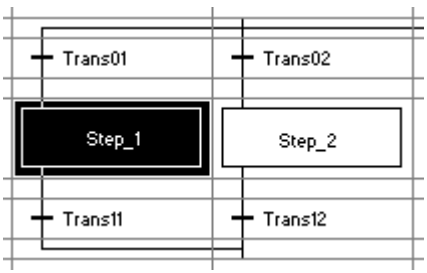
Delete element

→ *Edit* → *Delete element*



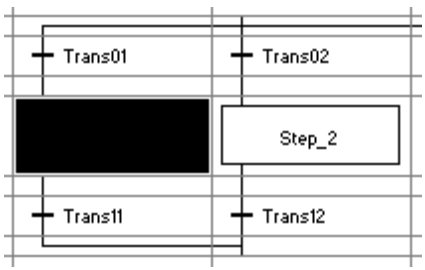
di1412uk.bmp

Every selected parameterizable element (*transition, step* or *goto*) can be edited.
The selected element is deleted after a safety interrogation.



di1460.bmp

Before deleting an element



di1461.bmp

After deleting an element

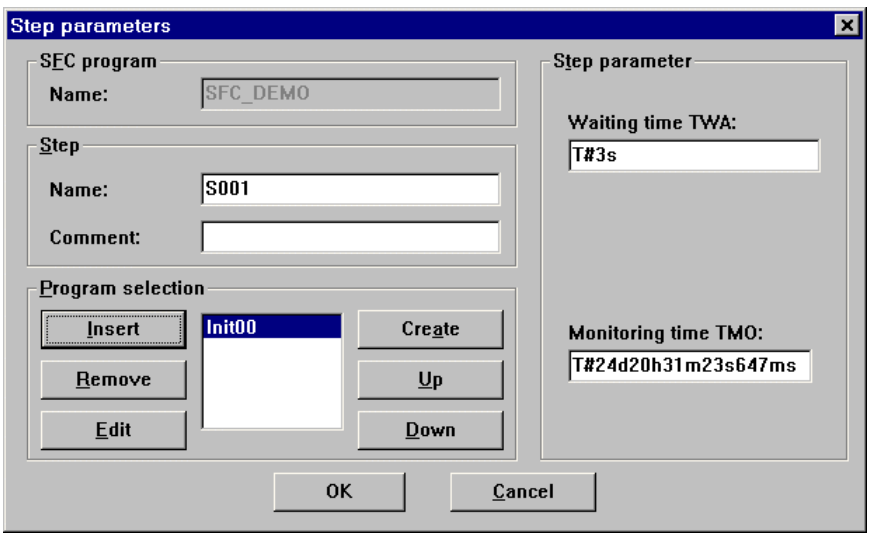
G 4.4 Parameters of elements

G 4.4.1 Step parameters

→ *Edit* → *Parameters of element*

Double click on step

If a step has been previously selected, the parameter definition mask for steps will be displayed:



di1413uk.bmp

SFC program

Name: Displays the name of the SFC program

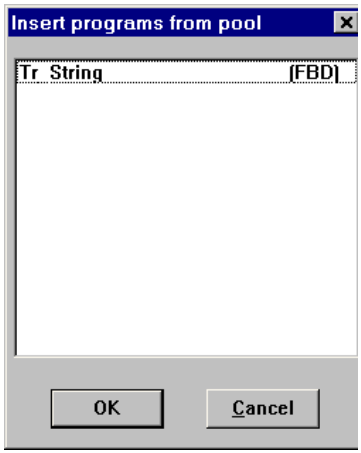
Step

Name: The step name can be edited.

Comment: The short comment of the step can be edited.

Program selection

INSERT



di1414us.bmp

FBD, LD or IL programs can be selected from the pool and edited when the selected step is active. The desired program is selected and inserted into the **program list** of the step with the OK button.

REMOVE — The selected program is removed from the step parameters and stored in the pool.

EDIT — The program selected with **select program** (colored envelope) is called up in the corresponding editor (IL, FBD, LD), so that e.g. changes can be entered. Before the editor is called up, however, the SFC program must be stored in the next interrogation.

CREATE — A new program can be stored. To do this, the desired type of program is selected in the **object selection** mask and inserted into the **program list** of the step with OK. To be able to continue editing in the corresponding editor, see → *Edit.*
The SFC program must be stored when changing to the IL/FBD/LD editor.

UP — The program selected with **select program** is placed in the **first position** of the processing sequence.

DOWN — The program selected with **select program** is placed in the **last position** of the processing sequence.

Step parameters

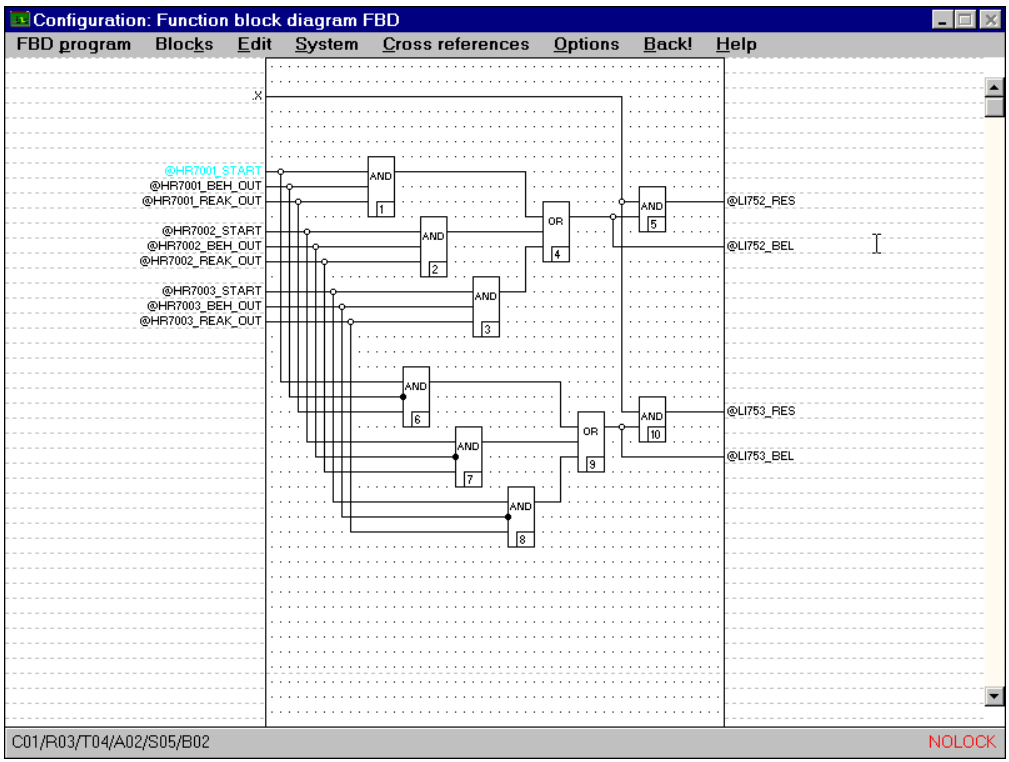
Waiting time TWA: The waiting time TWA is the minimum dwell time of the SFC program in one step. On activating Edit, subsequent transitions will be enabled only after expiry of the TWA.

☞ The wait time state can be polled within a step with the variable T.

Monitoring time TMO: The monitoring time TMO is the maximum desired dwell time in a step. On overshooting the TMO an alarm with the configured priority stage P1 - P5 (see **G-43, TMO message**) will be given.

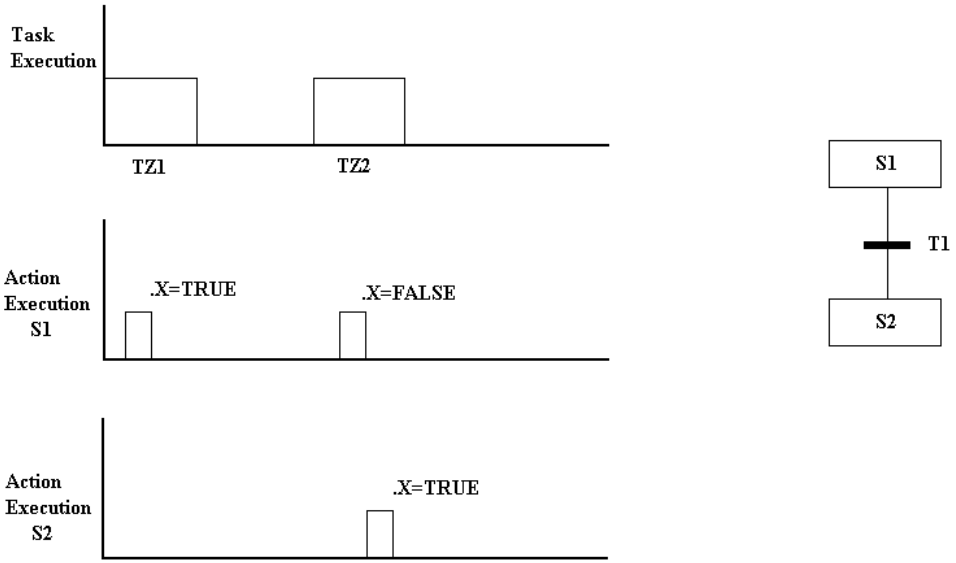
☞ After fulfilling the subsequent transition, an exact program cycle will be rerun. The flag **.X** is then set to 0. If actions are to be reset, this can be done with this flag.

G 4.4.2 Example of a step program



di1462uk.bmp

When a **transition** is successfully concluded, the **.RESULT** flag is set and the following **step** is computed cyclically until the transition flag of the next transition is also set to **.RESULT**. At the same time the step flag **.X** is set until the **.RESULT** flag is set in the next transition.



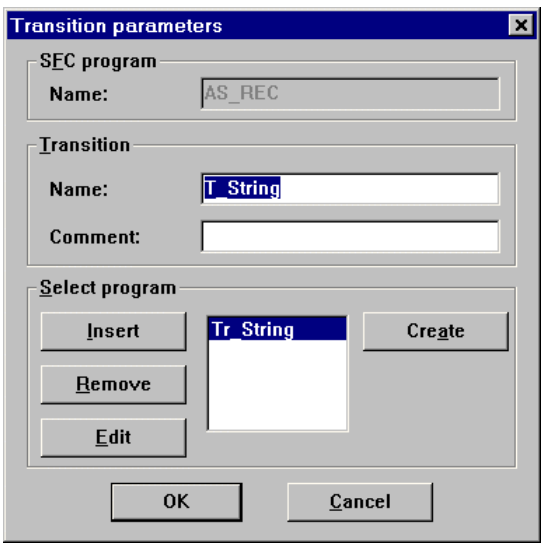
The example above shows how the steps are calculated after the transition is switched.
Step 1 is first calculated for a further cycle with .X = TRUE. After this there follows another cycle with .X = FALSE.
Step 2 is only calculated in the second cycle after the transition is switched, and in this cycle the calculation of step 1 is always completed before the calculation of step 2 commences. The system flag .X is also only set to TRUE in the second cycle.

G 4.4.3 Transition parameters

→ *Edit* → *Parameters of element*

Double click on transition

If a transition has been previously selected, the parameters will appear in the parameter definition mask for transitions:



di1416uk.bmp

SFC program

Name: Displays the name of the SFC program

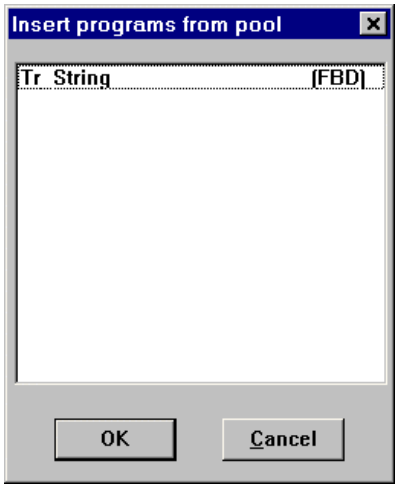
Transition

Name: Transition name can be edited

Comment: Short comment of the transition can be edited

Select program

INSERT



di1414uk.bmp

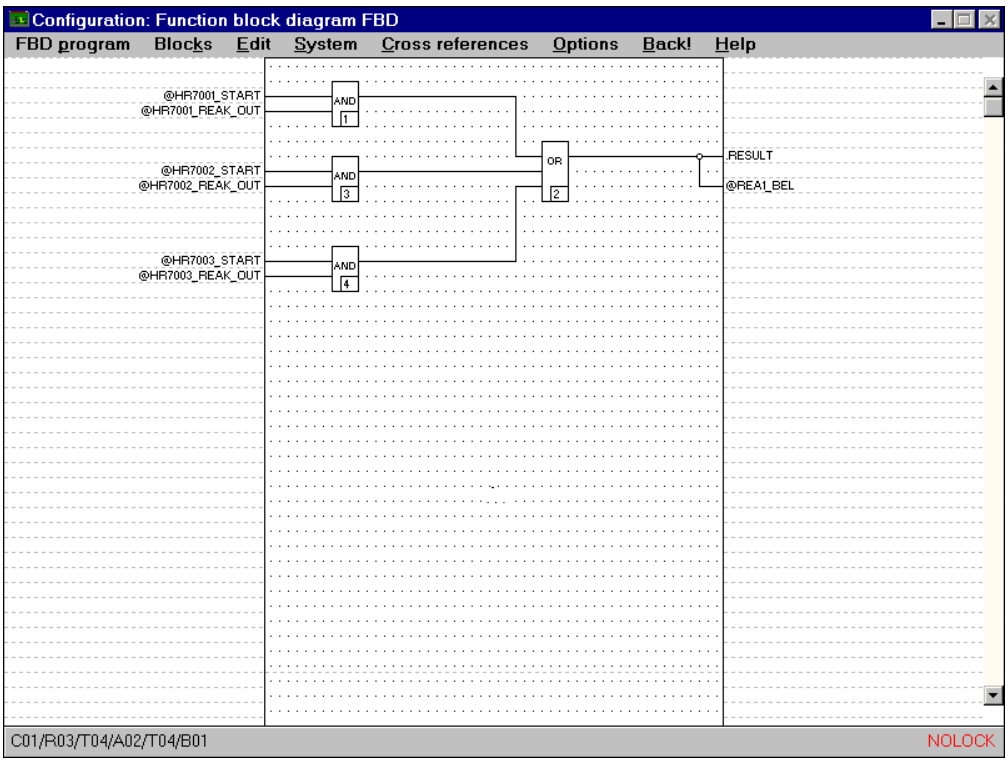
Only **one** program of the type FBD, LD or IL can be selected and edited as a transition when the transition in the sequence control is active. The desired program is selected and inserted in the program list with the OK button.

REMOVE The selected program is removed from the transition parameters and stored in the pool.

EDIT The program selected under **select program** (colored envelope) is called up in the corresponding editor (IL, FBD, LD), so that modifications for instance can be entered. Before the editor can be called up, however, the SFC program must be stored in the next interrogation.

CREATE A new program can be stored, when no other program has been inserted into the **program list** of the transition. To do this, the desired program is selected in the **object selection mask** and added to the **program list** of the transition with OK. To be able to continue editing in the appropriate editor see → *Edit*.

G 4.4.4 Example of a transition program



di1463uk.bmp

When a **transition** is successfully concluded, the **.RESULT** flag is set and the following step is computed cyclically until the transition flag of the next transition is also set to **.RESULT**. At the same time the step flag **.X** is set until the **.RESULT** flag is set in the next transition.

G 4.5 Edit program

→ *Edit* → *Edit program*

The same windows are called up as in *Parameters of elements*.
However, in the respective program list only the programs assigned to the step or the transition are displayed.
By clicking on EDIT the program corresponding to the chosen step or transition is displayed in the respective edit mode (IL or SFC).

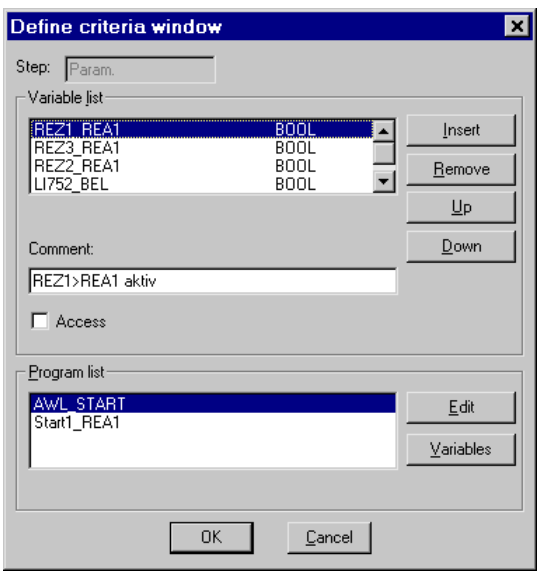
G 4.6 Define Criteria Window

Criteria windows can be defined for the operator interface, in order to give the user instructions for the current processing activity. Variables whose state or value can be later polled in DigiVis at the appropriate step can be selected. Variables can be entered on the one hand from the systemwide variables list, on the other hand, from the programs assigned to the step. To this end, however, the corresponding programs (FBD, LD or IL) must have been entered during parameter definition. Up to 20 variables of any data type can be entered. Variables of data type REAL, WORD or also INT can be newly written in the operator interface. *Access* ☑ must have been configured for this purpose.
A tag can also be assigned to each variable. This enables a relevant faceplate to be called up on the operator station in the operation dialog of the criteria window.
The criteria window for transitions is divided into three section **&&, >=1** and of any **data type**. Each variable of data type BOOL is marked later in color on the operator station, if a previously defined state has been assumed. The variables of the third section can be of any arbitrary type and are used for displaying values. All entered variables can be controlled by the operator.

G 4.6.1 Define Step Criteria Window

→ *Edit* → *Define criteria window*

If a step has already been selected, the **mask** for setting parameters for the step criteria window is displayed.



di1468uk.bmp

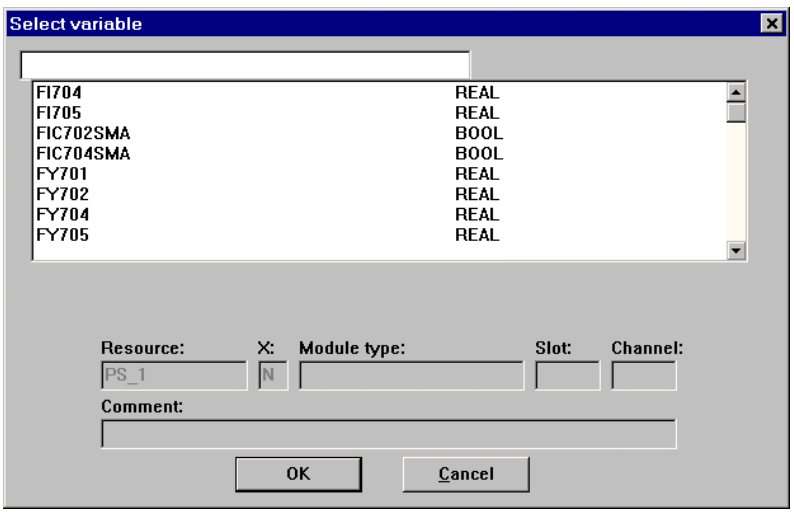
Step — Name of the selected step, cannot be changed here

Comment — Comment for the Criteria window, appearing later on the operator interface.

Access — ☑ Write (operator intervention) of the selected value is permitted.

Variable List

INSERT



di1469uk.bmp

A variable can be selected from the global variable list and taken over by clicking OK.

REMOVE — The selected variable is removed from the variable list without any prompt for confirmation.

UP / DOWN — These buttons can be used to define or alter the display order of variables in the associated criteria window for the sequential function chart. UP moves the selected variable one position upwards in the variable list, DOWN does the opposite.

ASSIGN TAG — These buttons can be used to show a selection list of the tags already configured in the system, in order to assign just one tag to the selected variable. This enables a faceplate to be chosen for the selected variable in the user interface (DigiVis). The tag's faceplate then makes direct operator action on that tag possible.

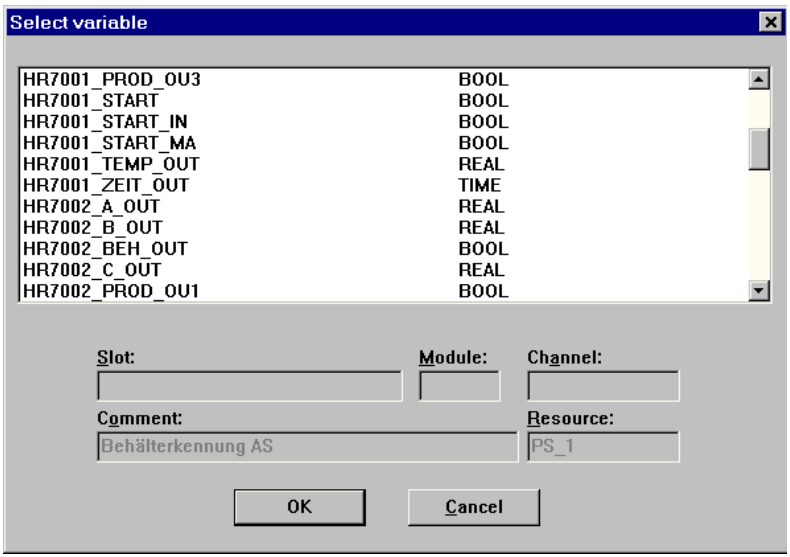
RELEASE TAG — The tag assigned to the variable is released again. It is thus not possible in the user interface to select any tag's faceplate for this variable.

Program List

EDIT

The program selected (highlighted) in the **program list** is called up in the appropriate editor (AWL or FBS) to enable, for example, changes to be made to it. Before the editor is called up, the program must be stored through the following dialog.

VARIABLES



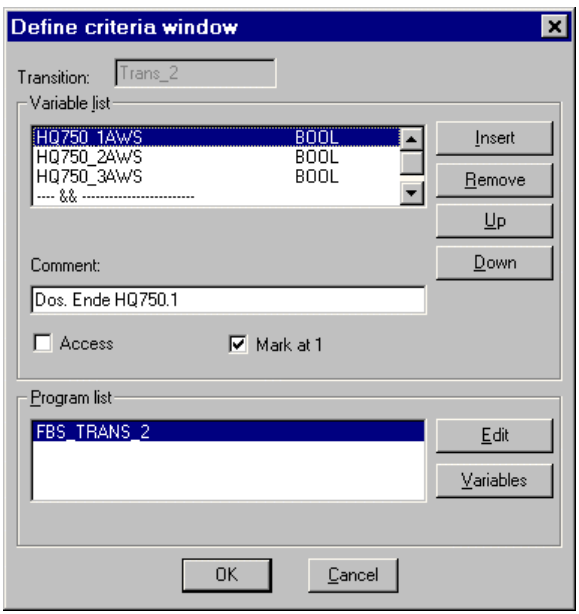
di1470uk.bmp

A list is displayed containing all the variables belonging to the program selected under the step's **program list**. Only **one** variable may be selected. Comment and resource name are also displayed. Where an input or output has been specified, the slot, module and channel are also displayed. When OK is clicked, the variable is copied across and inserted in the **variable list** in the criteria window.

G 4.6.2 Define Transition Criteria Window

→ *Edit* → *Define criteria window*

If a transition has already been selected, the **mask** for setting parameters for the transition criteria window is displayed.

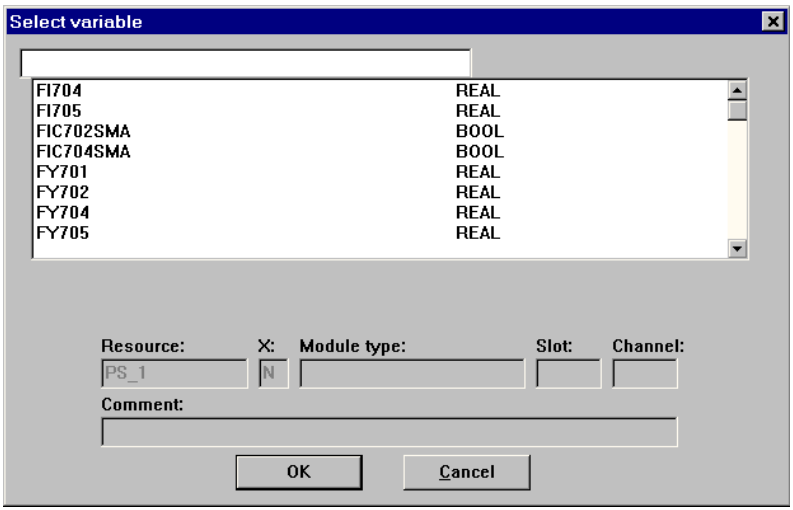


di1471uk.bmp

Transition Name of the selected transition, cannot be changed here

Variables List

INSERT



di1469uk.bmp

A variable can be selected from the global variable list and taken over by clicking OK.

REMOVE — The selected variable is removed from the variable list without any prompt for confirmation.

UP / DOWN — These buttons can be used to define or alter the display order of variables in the associated criteria window for the sequential function chart. Boolean variables can also be linked together with the operators **&, >=1**. UP moves the selected variable one position upwards in the **variable list**. DOWN does the opposite.

ASSIGN TAG — These buttons can be used to show a selection list of the tags already configured in the system, in order to assign just one tag to the selected variable. This enables a faceplate to be chosen for the selected variable in the user interface (DigiVis). The tag's faceplate then makes direct operator action on that tag possible.

RELEASE TAG — The tag assigned to the variable is released again. It is thus not possible in the user interface to select any tag's faceplate for this variable.

Comment — Comment for the Criteria window, appearing later on the operator interface.

Access — ☑ Write (operator intervention) of the selected value is permitted.

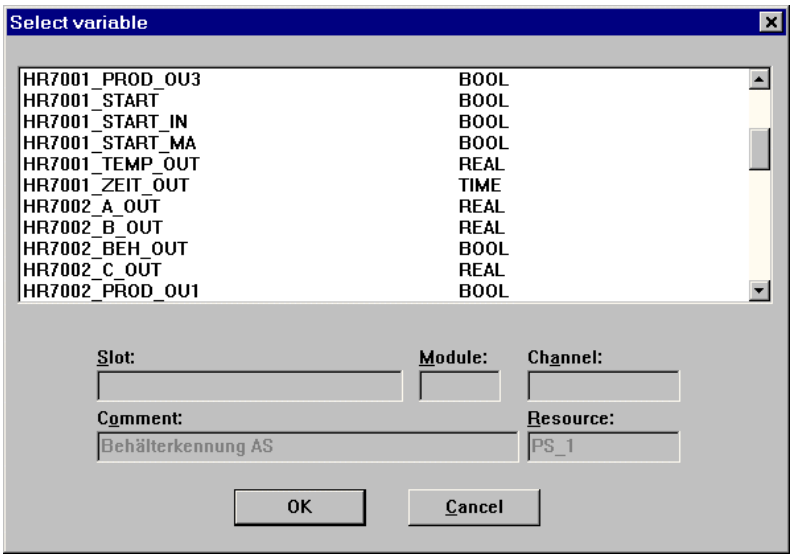
MARK AT TRUE — The selected variable's state can be inverted simply by clicking on it. Another click will reset the state.

Program List

EDIT

The program selected (highlighted) in the **program list** is called up in the appropriate editor (AWL or FBS) to enable, for example, changes to be made to it. Before the editor is called up, the program must be stored through the following dialog: YES, NO, CANCEL.

VARIABLES



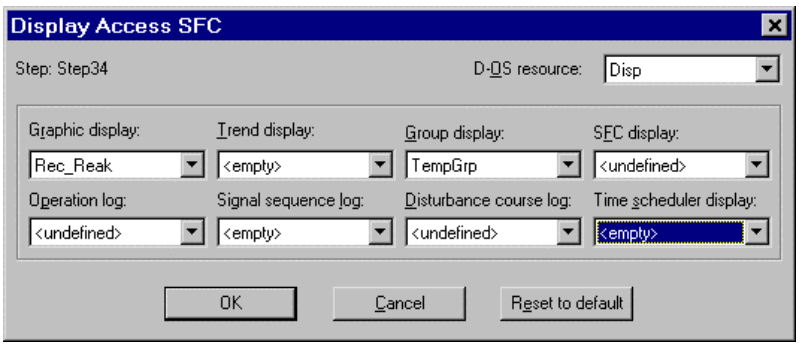
di1470uk.bmp

A list is displayed containing all the variables belonging to the program selected under the step's **program list**. Only **one** variable may be selected. Comment and resource name are also displayed. Where an input or output has been specified, the slot, module and channel are also displayed. When OK is clicked, the variable is copied across and inserted in the **variable list** in the criteria window.

G 4.7 Define display access

Displays and logs must be allocated for each transition and each step, and it should then be possible for the operator to call these via the DigiVis Operator dialog on the operator station. Here, either all displays or only those of one operator station are presented.

→ *Edit* → *Define display selection*



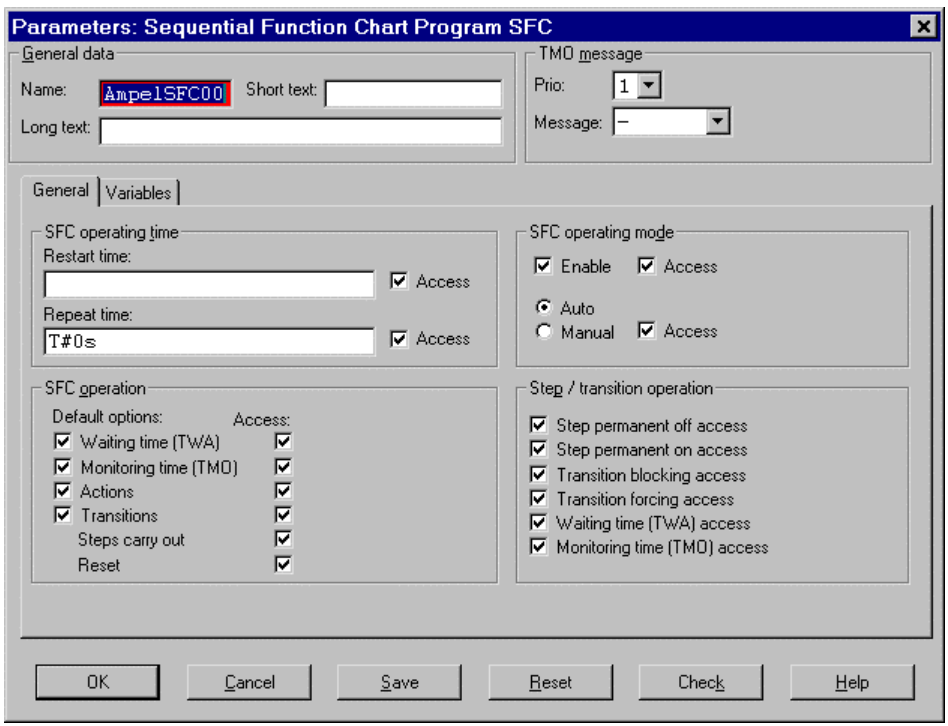
di1473uk.bmp

The selector lists show for the respective display or log type the corresponding project elements of all D-OS resources.

See also **Engineering Manual, Operator Station, Standard Displays.**

G 4.8 Parameters of SFC

→ *Edit* → *Parameters of SFC*



di1421uk.bmp

General data

Name This is the name of the SFC program. The name is entered in the tag list.

Short text This is the short text allotted to the SFC program. It can be up to 12 characters long. All characters are possible.

Long text This is the long text allotted to the SFC program. It can be up to 30 characters long. All characters are possible.

TMO message

Prio Alarm pertaining to the priority stage 1 to 5.

Message For sending configured messages.

Upon overshooting the monitoring time TMO, an alarm pertaining to the selected priority stage with related message text is generated at the process station.

SFC operating time

Restart time

Restart time is the time for restarting the sequence control. Contrary to the repeat time, the new start time represents just one time for starting the sequence control. Changing the new start time together with the repeat time also influences the time for a cyclical execution of the sequence control. By activating the operation mode *Access*, the change of the *Restart time* by the operator at the operator station is permitted.

Input format:
year-month-day-h:min:s.ms

Example:
DT#1994-12-31-23:59:59.999

Repeat time

Repeat time is the waiting time between two sequence control starts. Upon stating the repeat time and reaching the start time or new start time, the sequence control is executed. If the new start time is fixed, this is dominant via-à-vis the repetition time. If the repetition time is smaller or equal to the runtime of the sequence control, the sequence control is re-started immediately after completion of the sequence control.
By activating the operation mode *Access*, the change of the *Repeat time* by the operator at the operator station is permitted.
The input is effected according to IEC 61131-3 notations.

The set *Restart time* is dominant vis-à-vis the *Repeat time*.

SFC operating mode

Enable

By activating the operating mode *Enable*, the sequence control is enabled automatically when the start conditions are fulfilled.

Access

By activating the operating mode *Access,* user access on the operator station is allowed to be activated.

Auto/Manual

By activating the SFC operating mode *Auto*, transitions are enabled throughout the Program. With *Manual* the transitions are enabled by the operator of the operator station by means of DigiVis.

Access

By activating the operating mode *Access*, the changeover between *auto/manual* by the operator at the operator station is permitted.

SFC operation

Instructions for the execution of the SFC program on the process station are defined in this part of the parameter definition mask. With *Access* ☑ the accessibility of the respective options can be activated or deactivated by the operator at the operator station.

Waiting time (TWA) is the minimum dwell time of the SFC program in a step. Because of the activation of edit, subsequent transitions will be step enabled only after the expiry of TWA.

Monitoring time (TMO) is the maximum dwell time desirable in a step. If the TMO is overshot, an alarm with the configured priority stage P1 - P5 (see → **parameters of SFC program**) will be sent.

Actions If the step is active and is edit enabled, all actions assigned to the step will be edited. The actions assigned to the step will not be edited if the edit is not activated.

Transitions If the transition is active and is edit enabled, the transition is edited and the transition state checked. The transition will not be edited if the edit is not activated.

Steps carry out If *Access* is ☑, the operator at the operator station can positively switch all enabled transitions whose transition state has been fulfilled.

Reset With this operation mode activated, the operator at the operator station can reset all states of the SFC program.

Step / Transitions operation

Step permanent off access The actions of the steps are always suppressed. The step must be selected in the sequence control display.

Step permanent on access Actions are forced; compulsory execution.

Transition blocking access The execution of a transition is suppressed. The SFC interpreter runs to this transition and waits for an operation mode. Suppressing the transition is tantamount to setting a breakpoint in a program.

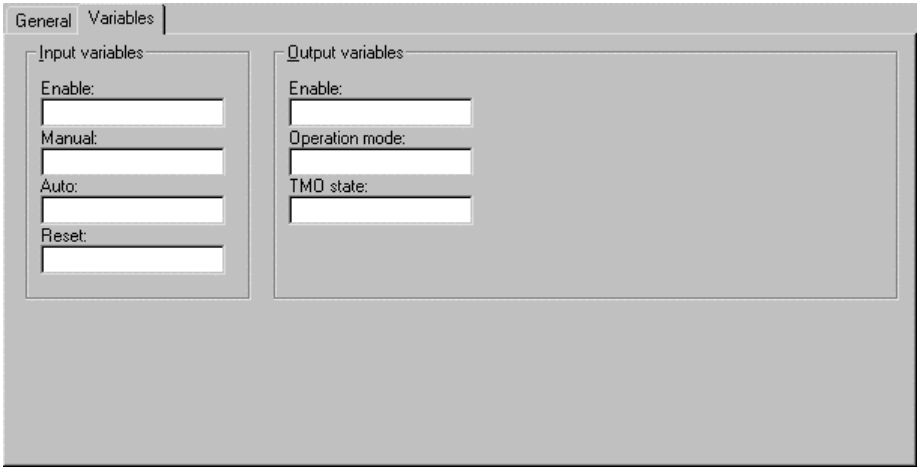
Transition forcing access If the transition is enabled, it can be forced and positively operated. The forcing of a transition is only valid for one run of the SFC interpreter on the process station.

Waiting time (TWA) access The waiting time in a step can be changed by the operator at the operator station.

Monitoring time (TMO) access The monitoring time can be changed by the operator at the operator station.

Variables of SFC program

→*Edit* → *Parameters SFC* →



di1472uk.bmp

Enable — Enable can be set for the SFC program via a freely configurable Boolean variable. The current Enable state can be routed to a further Boolean output variable. The Enable variable and the Enable operator parameter are interconnected via an OR function.

Manual/Auto — The operating mode can be set via two freely configurable Boolean variables. The same operating philosophy applies as for the controller function blocks. The variables are dominant vis-à-vis the modes operation parameter, the Manual variable is dominant vis-à-vis the automatic variables.

Operation mode — The current mode can also be routed to a freely configurable Boolean output variables. Automatic=TRUE.

Reset — Reset can be effected via a freely configurable Boolean variable. The reset variable and the reset operation parameter are interconnected acc. to an OR function. The reset signal is evaluated only in the manual mode.

TMO state — The state indicating that the monitoring time has expired on at least one step can be routed to a freely configurable Boolean output variable. This variable is then also set to TRUE if no TMO message has been configured.

G 4.9 Edit elements

Cut

 → *Edit* → *Cut*

If only elements without any parameters (steps / transitions) are selected, these will be removed from the working area and stored in a buffer memory (see also → *Insert*).

Is the selection at least a step or a transition, a window will appear to question, if the block should be deleted.

YES	Cut out and store elements in a buffer memory.
NO	Retain and store elements in a buffer memory (see also → *Insert*).

Copy

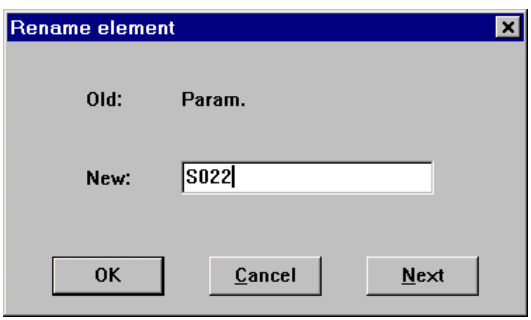
 → *Edit* → *Copy*

The selected elements are stored in a buffer memory and can now be inserted at another place.

Paste

 → *Edit* → *Paste*

Elements previously stored in a buffer (see also → *Cut* and → *Copy*), are inserted at the cursor position.
If steps or transitions are inserted, provide them with new names. To do this, insert a window for each of these elements so that names can be changed.



di1424uk.bmp

NEW	Insert new name
OK	Element is inserted at the appropriate place with the new name. The parameters of the new element have been reset.
CANCEL	The entire insert action is interrupted.
NEXT	The displayed element is not inserted. The next element is presented for renaming.

If there is inadequate space for the insert at the marked point for all elements, the following message will be given:

di1425uk.bmp

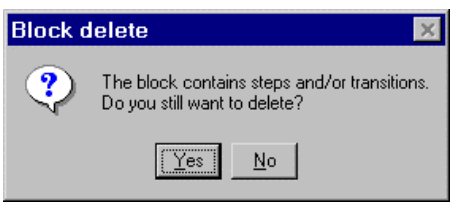
By acknowledging this message with OK , the block can be positioned anew with the left mouse key pressed. Cancel the function by either pressing ESC or the right mouse key.

After cutting the parameterized elements, their parameters are reset after the insert, i.e. inserted programs become available again in the POOL.
If the parameter definition should be retained, do not shift it with *Cut - Paste* (see **page G-22, Shift blocks**).

Delete

→ *Edit* → *Delete*

Delete selected elements. If steps or transitions are contained in the selection, a window will appear with the question if the block should be deleted.



di1422uk.bmp

G 4.10 Export and import blocks

The export and import of blocks permits re-utilization of project parts in the existing project or in other projects.

Export block

→ *Edit* → *Export block*

This is used to export the entire contents of the selected block in ASCII into an **.as-file** which can be re-imported with the menu item *import block*. Assign the file name in the open window "Sequential function chart export".

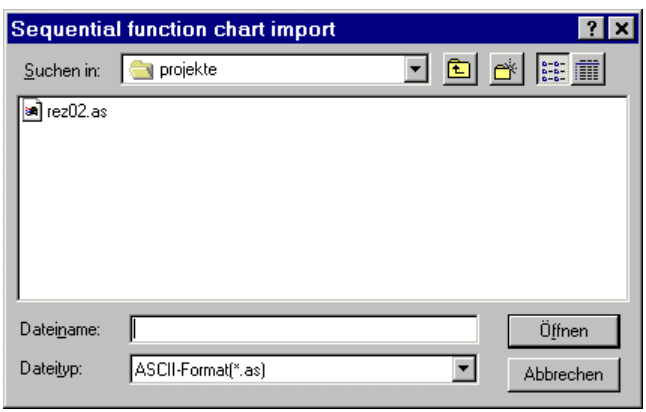
Access to the directory is preset by default. The last active directory will be called up. The directory for export files is on the hard disk c:\freelance\export.

Import block

→ *Edit* → *Import block*

This is used to import the contents of a block from an **.as-file** which had previously been generated with export block to the preselected position in the SFC program.

Access to the directory is preset by default. The last active directory will be called up. The directory for import files is on the hard disk c:\freelance\export.



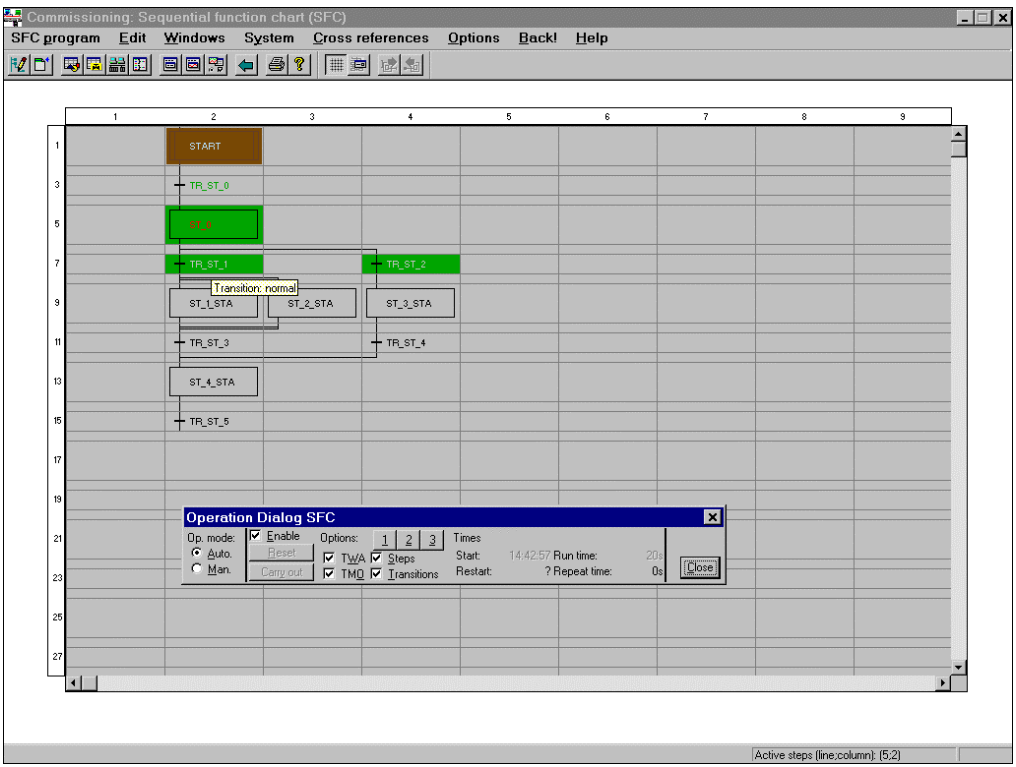
di1427uk.bmp

G 5 Commisioning the Sequential function chart (SFC) program

While commissioning an SFC program it is possible for the program to be operated using functions similar to those available in an SFC display at an operator station.

From commissioning it is thus possible to

- Switch the SFC program between automatic and manual.
- Release or block the SFC program.
- Display the current state of the step or transition.
- Execute all active transitions and steps manually once.
- Execute steps in manual mode and, in contrast to DigiVis, in automatic mode also. The processing of steps or transitions is controlled by the *Options* setting.
- Change time parameters, such as restart time, repeat time and wait time.
- Switch steps permanently on or off.
- Block or force transitions on a one-time basis.



di1581uk.bmp

The individual steps and transitions are displayed depending on their state and the type of action execution in the display area of the SFC program. The section of the display can be moved using the horizontal and vertical scroll bars.

During the processing of a sequential control program, no more than 8 steps may be active simultaneously.
The background color in the display area depends on the operating mode. In **Automatic mode** the background is **transparent** and in **Manual mode** it is **blue**. The display of steps and transitions is the same in both modes.

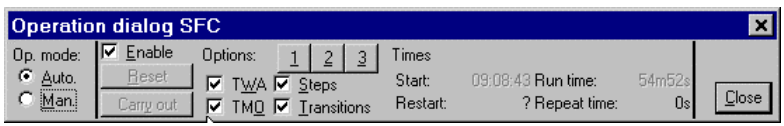
In contrast to DigiVis, it is also possible to change the state of steps and transitions in automatic mode.

In commissioning, it is not possible to change the structure of the SFC program. This is only possible in configuration.

Every time a sequential control program becomes overloaded, it will, as a result, be restarted with the initial step.

G 5.1 Operation dialog SFC program

Operation → SFC program...



di1552uk.bmp

Settings entered within the operation dialog are valid for the entire SFC program. The operation dialog is broken down into the areas of **Operating Mode, Options** and **Times**.

Operating mode
In the **Auto** operating mode the transitions are stepped through by the program. In the **Manual** operating mode transitions and steps can be activated by the operator.

Step and transition execution

Enable Allows the execution of the SFC program. If enable is activated and the new restart time or repeat time has been reached, the initial step of the SFC program is executed.

☞ The enabling of the SFC program is independent of whether the operating mode is auto or manual.

RESET The SFC program in the process station is reset to the initial step.

☞ Reset is only possible in manual mode!

CARRY OUT All transitions in the enabled state whose transition criteria are fulfilled, are stepped through once.

☞ Global step is only possible in manual mode

TWA If this field is checked, the **minimum wait time** ***(****TWA)* for all steps in the SFC program is monitored.

TMO If this field is checked, and one is in manual operating mode, the respective **monitoring times** ***(****TMO)* of the active steps are monitored. In auto operating mode monitoring always takes place.

Steps If this field is checked, then the respective actions of the active steps are executed.

Transitions If this field is checked, the programs which are linked to the transitions are executed. The transition criteria are tested.
If this field is not checked, the programs which are linked to the respective transitions will not be executed, and the transition criteria are always taken as fulfilled.

Options: With the activation of one of three buttons, it is possible to set a predefined profile for the processing of actions and transitions e.g. for the monitoring of the times TWA and TMO

TWA, TMO, actions and transitions are not activated.

Actions activated.

Actions and transitions activated.

Times in the SFC program
The times in the global operation dialog are valid for the entire SFC program. The start time and run time cannot be changed!

Start: The activation time of the initial step of the SFC program is defined as the **start time**. With each new run, the actual time is entered in the process station.

Runtime: The **run time** is the elapsed time since the start. The run time is reset to 0 s when the initial step is repeated.

Restart time The **restart time** is the time for a new start of the SFC program. In contrast to the repeat time, the restart time represents a single point in time for restarting the SFC program. When the restart time is changed in connection with the repeat time, the time point for the interval processing of the SFC program is influenced.

Repeat time The repeat time is the minimum wait time between two starts of the SFC program.
If the restart time is fixed, it will take precedence over the repeat time. If the repeat time is less than or equal to the run time of the SFC program, then the SFC program is started again immediately after ending.

Changing the restart or repeat time

 Position the cursor on the Restart time → Double click the left mouse button

Changes are made by entering the new value from the keyboard. The entry must be in IEC 1131 - 3 date and time notation.

The restart time is entered in the Date and Time format (DT):

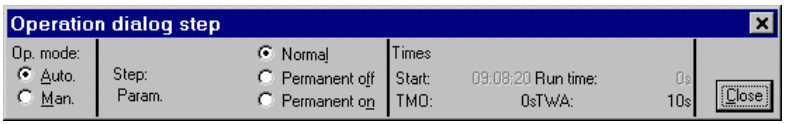
Example: DT#1999-12-31-23:59:59.99

The repeat time is entered in time format (TIME):

Example: T#3m30s

G 5.2 Step operating dialog

Select step with left mouse button → *Operation* → *Step*...



di1358uk.bmp

CLOSE The operation dialog step is closed and the global SFC operation dialog reappears.

Operating mode
In the **Auto** operating mode, the transitions are stepped through by the program. In the **Manual** operating mode, transitions and steps can be activated by the operator.

Action execution
This section of the operation dialog is used to fix the action execution of a step.

Normal The step is processed normally.

Permanent off The step is never processed.

Permanent on The step is always processed.

Times
The times in this dialog can only be seen for a single step.

Start: The **start time** shows the beginning of the execution of the selected step. With each new execution of a step the start time is updated.

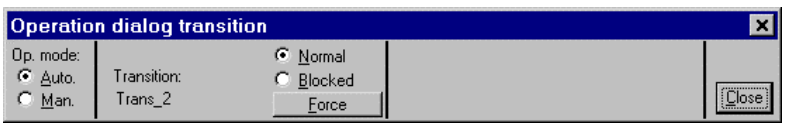
Run time: The **run time** shows the time that the active step has been active. With each new execution of a step, the run time is reset to 0s

TMO: **Monitoring time** for this step. If this time is exceeded, a message is generated.

TWA: **Minimum waiting time** for one step.

G 5.3 Transition operation dialog

Select transition with left mouse button →*Operation* → *Transition*...



di1359uk.bmp

CLOSE — The operation dialog for transitions is closed and the global operation dialog reappears.

Operating mode
In the **Auto** operating mode, the transitions are stepped through by the program. In the **Manual** operating mode, transitions and steps can be activated by the operator.

Transition criteria
This section of the operating dialog is used to influence the manner in which the selected transition is made. In the **Auto** operating mode, transitions are always made in the **Calculated** mode**.**

Normal — The transition is made in the calculated mode.

Blocked — The transition is *blocked*. The transition is not made even if the transition criteria are fulfilled.

FORCE — Immediately after the transition is enabled, the transition is made, independent of the transition criteria.

G 5.4 Step states

Steps in the Freelance 2000 system can have the states **inactive**, **active** and **was active.**

inactive — A step is **inactive** when it is not run through during a cycle. While a step is inactive, the programs linked to it are not executed.

active — A step is set to the **active** state when the transition criteria of the previous transition have been fulfilled. Once a step is active, programs linked to it, are executed.

was active — After a step has been run through during a cycle, the step goes from the active state to the **was active** state**.**

faulty — The monitoring time of a step has been exceeded.

G 5.5 Step action execution

The Freelance 2000 system offers the user the modes **normal, permanent off** and **permanent on** for the execution of step-related actions.

☞ Action execution is independent of step state.

normal If a step is active, the actions related to that step are executed.

permanent off The actions related to that step are never executed.

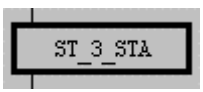
permanent on The actions related to that step are always executed.

G 5.6 Display of steps in the SFC program

The appearance of steps in the SFC program display depends upon the step state and the action execution mode.

tg014.bmp

initial step

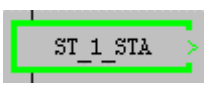


tg015.bmp

normal step

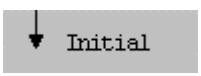
tg016.bmp

permanent off step



tg017.bmp

permanent on step



tg018.bmp

jump out

tg019.bmp

jump in

Step state	Symbol section	Action execution		
		normal	permanent off	permanent on
inactive	Background	gray	gray	gray
	Lines	black	dark blue	signal green
	Text	black	black	black
active	Background	dark green	dark blue	signal green
	Lines	black	black	black
	Text	white	white	black
faulty	Background	dark green	dark blue	signal green
	Lines	black	black	black
	Text	red	red	red
was active	Background	gray	gray	gray
	Lines	black	dark blue	signal green
	Text	dark green	dark green	dark green

Table 1: Colors used to display steps depending on their state and action execution mode.

G 5.7 Transition states

In the Freelance 2000 system, transitions can take on the states **not enabled, enabled**, **clearing** or **disabled**.

not enabled	Not all preceding steps have been active → transition criteria not evaluated
enabled	All preceding steps have been active → transition criteria evaluated
clearing	The transition criteria are fulfilled. All preceding steps become inactive and all dependent steps become active - transition is made.
disabled	All dependent steps are active - transition has been completed.

G 5.8 Display of transitions in the SFC program

The appearance of transitions in the SFC program display depends on their state.

di1355uk.bmp

enabled or **clearing**

di1356uk.bmp

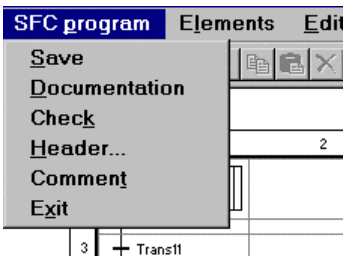
not enabled or **disabled**

Transition state	Symbol section	Execution of the Transition criteria		
		normal	**blocked**	**forced**
not enabled	**Background**	gray	gray	gray
	Lines	black	dark blue	signal green
	Text	black	black	signal green
enabled	**Background**	dark green	dark blue	signal green
	Lines	black	black	black
	Text	white	white	black
fulfilled	**Background**	gray	gray	gray
	Lines	black	dark blue	black
	Text	dark green	dark blue	dark green
completed	**Background**	gray	gray	gray
	Lines	black	dark blue	black
	Text	dark green	dark blue	dark green

Table 2: Colors used to display transitions depending on their state.

G 6 General Editing Function

→ *SFC Program*



di1403uk.bmp

G 6.1 Save

→ *SFC Program* → *Save*

All entries and changes made in this SFC program are saved.

G 6.2 Documentation

→ *SFC program* → *Documentation*

Exit the SFC editor for the documentation library. Here the project documentation is defined to suit user requirements and printed out.

See also **Engineering Manual, System Configuration, Documentation.**

G 6.3 Check of program elements

If something was overlooked, forgotten or wrongly entered during program input, the plausibility check returns the formal warning and error messages with the block number, so that corrections can still be made before commissioning The program can only be downloaded into the process station and commissioned if a plausibility check has run without error messages. The check is called from the *SFC program* menu:

 Select program element →*SFC program* → *Check*

All function-relevant entries are checked for syntactical and contextual correctness. Any errors and warnings are shown in the error list. If errors are detected by the check, the processing state of the program element features the **incorrect** state.

 Any newly entered, copied or moved program elements have the **incorrect** state.

G 6.3.1 Jumping to error locations after the plausibility check

After a new plausibility list has been created via *Plausibility check* or *Plausibility check all*, any errors that were detected are displayed to the user in a list box.

 Double-clicki the left mouse button

 Mark message → ENTER

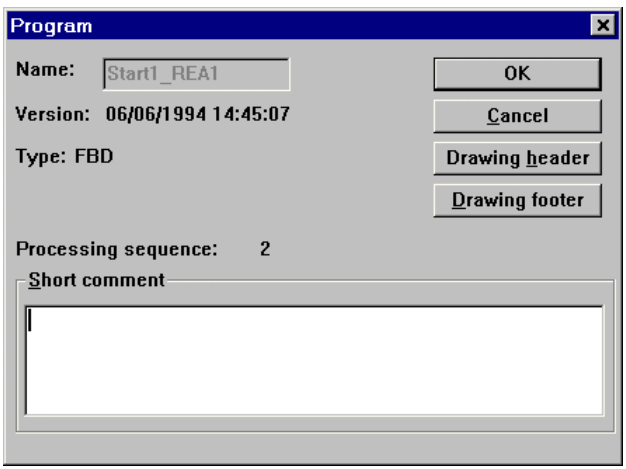
A jump to the field causing the error is executed.

After returning to the project tree and calling *Display errors*, the next plausibility message in the list will appear selected.
The help text relating to the marked message can be called up via a [Help] button.
The destinations of these jumps are exactly the same whether the plausibility check was called up in the project tree or in the program. If a jump is performed to a program page in which a selection was available previously (only possible after running plausibility check within an editor), then the selection will be lost in the process.

See also **Engineering Manual, System Configuration, Project tree, Plausibility check**.

G 6.4 Edit header

→ *SFC program* → *Header*



di1404uk.bmp

A program-specific short comment on the program documentation header can be entered or edited.
All colored fields for this SFC program can be edited in the drawing footer and header. See also **Engineering Manual, System Configuration, chapters Project Tree** and **Documentation**.

G 6.5 Comment

→ *SFC program* → *Comment*

Here, give a longer program-specific comment to describe the functionality.
See **Engineering Manual, System Configuration, Project Manager, Editing a project comment**.

G 6.6 Back!

 → *Back!*

The SFC editor is exited and the previous active program invoked.
If any modifications were made on the SFC program, an interrogation will appear on exiting, if these should be stored (YES / NO / CANCEL).

G 6.7 Hardcopy

 → *Options* → *Hardcopy*

The contents on screen are printed out.

G 6.8 End the SFC program

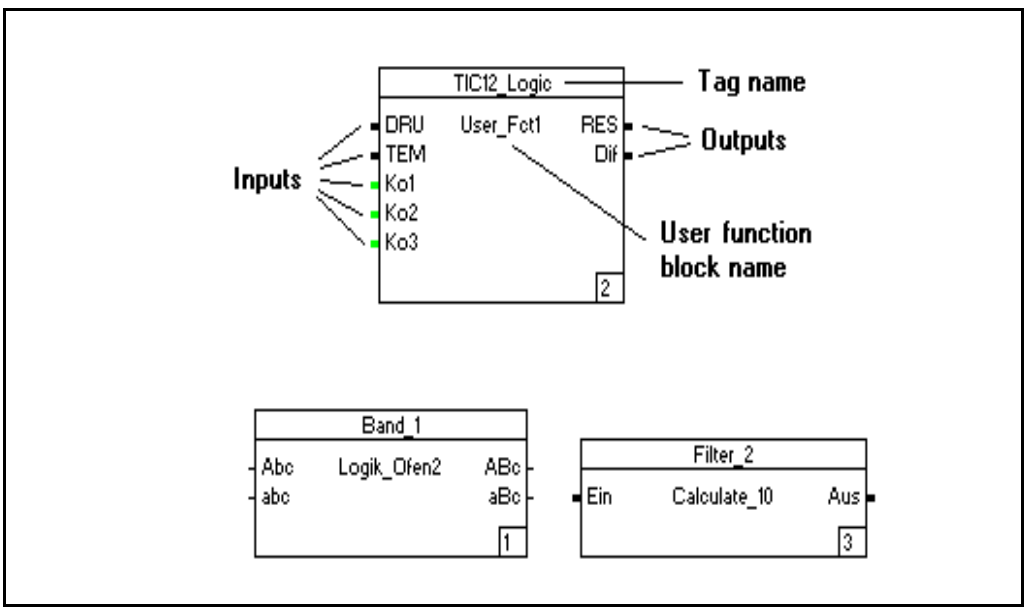
 → *SFC program* → *Exit*

End the SFC editor and return to project tree. In some cases, an interrogation will appear:

The present program has been changed.
Do you want to save the change?

→ YES → NO → CANCEL

H User Function Blocks



Contents

H 1 General Description - User Function Blocks

The **user function block (UFB)** facility makes it possible for users to create their own function blocks. This means that function blocks can be designed to meet the specific needs of particular sectors.

In working with user function blocks, **classes** and **instances** will be differentiated.

The specifications of a **user-created function block class** will determine the functionality and the appearance of a function block. As such, the class encompasses the user-created program in its entirety, with its functions, function blocks and variables, as well as the faceplate and the parameter mask.

The configuration of a user function block class is effected in the project tree under *Function Block Pool P-FB*. Every user function block class receives a class name which can be freely chosen and with which the block can be called from inside other programs.
The user function block program can be written in function block diagram (FBD), ladder diagram (LD) or instruction list (IL). The rules of the language used govern the structure of inputs and outputs, positioning, parameterization, etc.
Only after passing the plausibility check in the project tree, can a user function block be made available. It can then be found for later use under the *Blocks* → *User FB* menu.

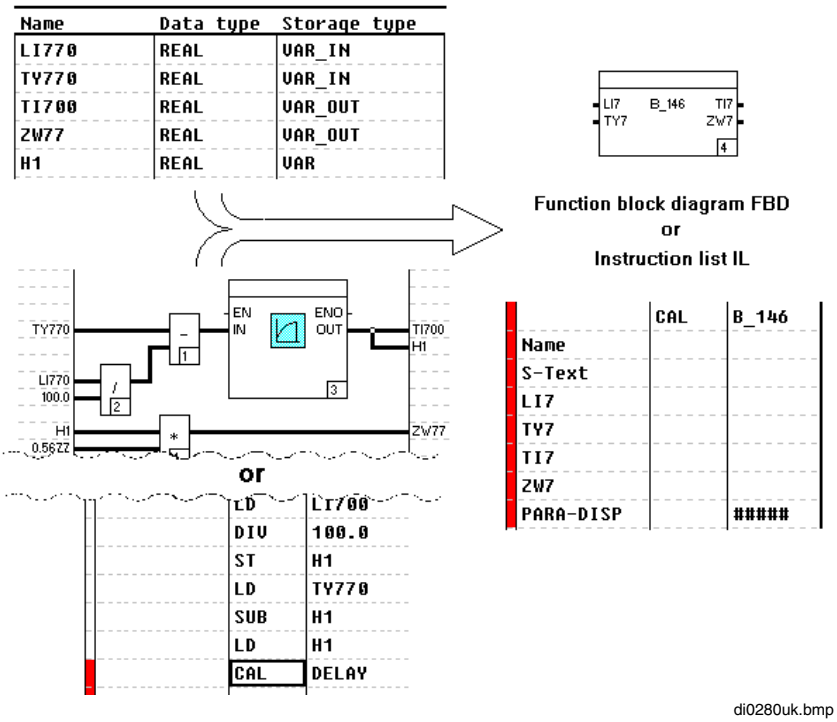
For actual use, **instances** of a user function block class must be created. Each instance has a filed-in parameter mask, which contains a tag name, short text and long text. Every user function block instance must be assigned one and only one tag name.
User function blocks can be called from all programming languages: IL, LD, FBD and SFC.

Changes to user function blocks are made to the function block class and effect all function block instances of that class in place. If new pins are added to a block, the function block instances must be replaced and the new pins accounted for.

User function blocks can be locked by the user with a password. They then appear only in their external representation: function blocks nested within them are no longer visible.

The **faceplate of a user function block** is created in the faceplate editor. The faceplate editor offers the full functionality of the graphic editor.

Schematic representation of user-defined function block



di0280uk.bmp

Creation of a user-defined function block:

- In project tree, open the **function block pool, P-FB**
- Create a **user function block class** in the pool
- With the Interface editor, enter the variable names for the function block class - **User FB variables**
- Then, choose creation of a user function block program in IL, LD or FBD
- Create the program
- Then, choose creation of a user function block faceplate
- Create the faceplate
- Save, then run the plausibility check in the project tree
- The user function block so created is callable from other programs under *User blocks*

H 1.1 User function block - classes and instances

To create a user function block, the user first creates the user function block class. Only as a next step can he or she create instances of this class.

A user function block class encompasses the full functionality and appearance of the function block. The information required is entered with the interface editor, the program editor and the faceplate editor.
A class itself cannot be run in a process station. For execution, an instance of the class must first be created. There can be as many instances of a class as desired.
An instance is the executable form of a class. Different instances are identified by their tag names. Each instance works with values specific to that instance (local variables and parameters).
In a function block instance, all local and output variable values are retained from one execution to the next. This means that the function block instance has an internal state, with the result that the same inputs need not always result in the same outputs.

Any user function block which has already been declared can be used in the declaration of another user function block.

H 1.2 Generation of user function block pool

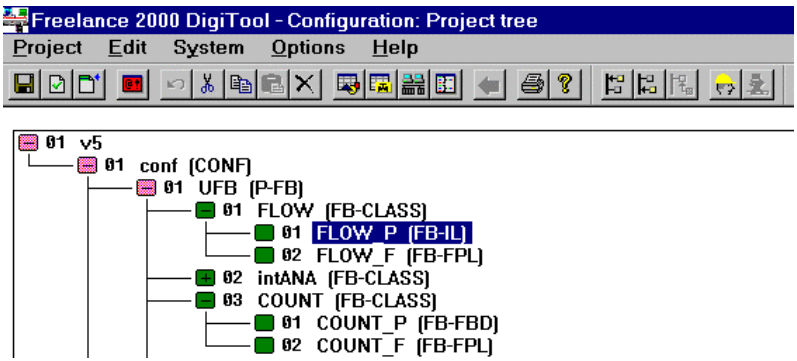
→ In project tree, go to the (CONF) level
→ *Edit* → *Insert next level*
Object selection window appears
→ Choose *Function Block Pool P-FB*
→ Enter pool name, max. 4 chars.
Name entered appears in project tree

The user function block pool must be created with a name in the project tree. Within it, an unlimited number of user function block classes can be declared.

Only one **user function block pool** may be created per project. All user function blocks are present in this pool.

H 1.3 Create a user function block class

→Select pool name (P-FB) → *Edit* → *Insert next level*,
Object selection window appears
→Choose user function block class
→Enter class name
Class name entered appears under the pool (P-FB) in project tree



di0281uk.bmp

Under P-FB the individual function block classes are shown by name as objects in the project tree. A user function block class name may be a maximum of 12 characters long, must follow the Freelance 2000 naming conventions and must be unique on a project-wide basis.

Class names should be chosen in such a way that collisions with possible available classes are avoided during import in other projects. The same basic principle applies to the allocation of tag names when defining the UFB class.

H 1.4 Create a user function block program

→Select class name (P-FB) → *Edit* → *Insert next level*
Object selection window appears
→Choose either FBD, IL or LD program
→Enter program name
→Program name entered appears under the user function block class in the project tree

The program to be executed as a user function block consists of one and only one FBD, LD or IL program. A combination of programs of different types is possible by nesting user function blocks inside one another.

The user function block program is created as described in **the Engineering Manual, System Configuration, chapter Project tree**.

H 1.5 Create new user function block faceplate

→ Select class name (P-FB) → *Edit* → *Insert next level*
Object selection window appears
→ Choose *User faceplate*
→ Enter faceplate name
→ Faceplate name entered appears under the user function block class in the project tree

Instance-specific values of a user function block instance can be displayed in DigiVis with the faceplate.
User function block faceplates are created with the faceplate editor. The faceplate editor offers the full functionality of the graphic editor.

Each user function block has one and only one faceplate.

H 2 Definition of User Function Block Classes

A user function block class is made up of the following components
- Interface
- Parameter dialog
- Text list
- Program
- Faceplate

H 2.1 Definition of user function block interface

H 2.1.1 Interface editor

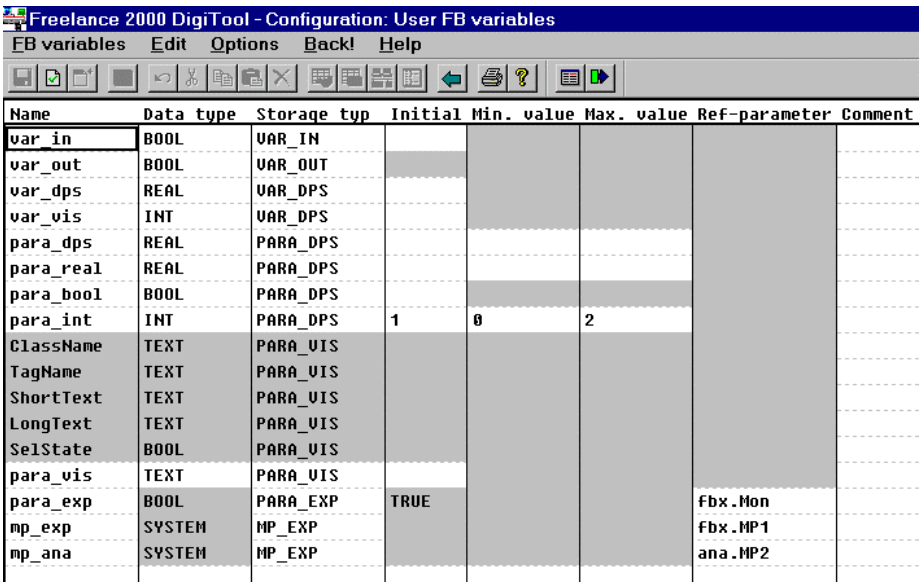
The variables used in the user function block program must be entered in the **Interface editor** under **User FB variables.** The parameter mask is created and the text list administered in the **Interface editor.**


→ **Project tree** → Double click on the node of the user function block class
→System → *User FB variables*

or


→ **Project tree** → select user function block class
→ *Edit* → *Program*
→System → *User FB variables*


→ **Configuration: function block diagram FBD**
→ **Configuration: instruction list IL**
→ **Configuration: ladder diagram LD**
→ *System* → *User FB variables*



Name	Data type	Storage typ	Initial	Min. value	Max. value	Ref-parameter	Comment
var_in	BOOL	VAR_IN					
var_out	BOOL	VAR_OUT					
var_dps	REAL	VAR_DPS					
var_vis	INT	VAR_DPS					
para_dps	REAL	PARA_DPS					
para_real	REAL	PARA_DPS					
para_bool	BOOL	PARA_DPS					
para_int	INT	PARA_DPS	1	0	2		
ClassName	TEXT	PARA_VIS					
TagName	TEXT	PARA_VIS					
ShortText	TEXT	PARA_VIS					
LongText	TEXT	PARA_VIS					
SelState	BOOL	PARA_VIS					
para_vis	TEXT	PARA_VIS					
para_exp	BOOL	PARA_EXP	TRUE			fbx.Mon	
mp_exp	SYSTEM	MP_EXP				fbx.MP1	
mp_ana	SYSTEM	MP_EXP				ana.MP2	

Legend:

:= May be edited

:= May not be edited, or no reasonable entry possible

di0283uk.bmp

The individual entries can be selected with a double click or using the menus. Entries can be made directly in the **Name**, **Initial value, Min. value, Max. value** and **Comment** fields. The **Data type, Storage type** and **Ref.-parameter** fields can only be filled in using the pop-up windows that appear.

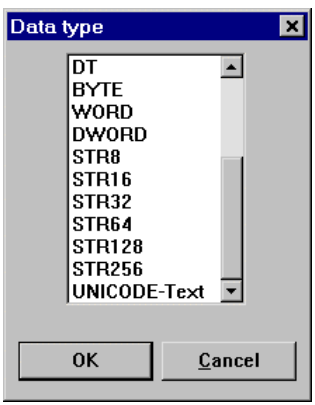
Name	Freely chooseable variable name. Conventions for the naming of variables apply.
Data type	A window with the different data types is opened with a double click. Choose desired data type and confirm with OK.
Storage type	The storage type selection window is opened with a double click. Choose desired storage type and confirm with OK.
Initial value	Enter a starting value using the appropriate data format.
Minimum value	Enter a minimum value using the appropriate data format.
Maximum value	Enter a maximum value using the appropriate data format.
Reference parameter	The reference parameter window is opened with a double click. Select the desired parameters and confirm with OK.
Comment	Any desired comment text, 32 characters maximum.

Name

For values with storage types **VAR_IN** and **VAR_OUT,** the first three characters of the name become the pin identifier. Upper or lower case are both allowed and are differentiated. If the first three characters of two pins are identical, a plausibility warning results. The order in the declaration follows the displayed pin layout in the user function block instances

☞ All defined variable names are valid only within the user function block class in which they were defined.

Data type



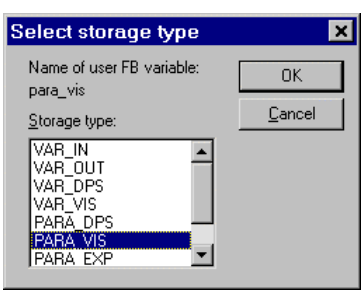
th001us.bmp

All Freelance 2000 data types are allowable as user function block variables. User-defined data types cannot be used.

There is a new data type for display in user faceplates that can be used inside user function blocks: **UNICODE-Text**. This data type makes it possible to enter texts in different languages. Message points are displayed with **SYSTEM** data type.

☞ The data type of variables with the PARA_EXT storage type depends on the data type of the nested function block.

Storage type



th002us.bmp

Every user function block variable has a storage type. The storage type determines how the variable is used inside the user function block. The storage type determines where the runtime value of the variable is to be found.

There is a basic distinction between the VAR_... and PARA_... storage types. VAR_... Storage types are used for internal processing. They do not participate in configuration, while PARA... Storage types do appear in the parameter dialog during configuration of the user function block instances.

VAR_IN represent inputs to the user function block. VAR_IN variables cannot be written to the user function block program. They can be displayed in the faceplate. At runtime, the user function block instance will be supplied at the cycle rate with values from the associated signal lines for further processing.

VAR_OUT represent the outputs of the user function block. They can be changed in the user function block program and can be also be displayed in the faceplate. At runtime, signal lines associated with VAR_OUT variables are updated at the cycle rate.

VAR_DPS are local variables used by the user function block running on the process station. They are used to hold intermediate values. VAR_DPS variables can be read from the faceplate. They are used for internal calculation in the faceplate.

PARA_DPS variables are used in the configuration of values that effect the processing at the process station, for example, operating mode switching. They can be read and written from the faceplate. PARA_DPS variables are commissionable—i.e. they can be written to or corrected in commissioning mode.

PARA_VIS variables are used in the configuration of variables which are only used in the faceplate, such as instance-specific display text, and operation locking. PARA_VIS variables cannot be changed from commissioning mode.

PARA_EXP variables are used to reference data of nested function blocks (standardized or user function blocks). They inherit their other characteristics from the parameters they are used to reference. Each variable with the PARA_EXP storage type can be used to reference one and only one parameter of a nested function block.

MP_EXP variables are used to reference message data from nested function blocks. Each one references a complete message structure, comprising message type, message priority, hint data and message text.

Storage type	Data source	(1)	(2)	(3)	(4)	(5)	(6)	Use
VAR_IN	D-PS	x	x	-	-	-	-	Input pin
VAR_OUT	D-PS	x	x	x	-	-	-	Output pin
VAR_DPS	D-PS	x	x	x	-	-	-	Internal D-PS variable
VAR_VIS	DigiVis	x	x	-	-	-	-	Internal DigiVis variable
PARA_DPS	D-PS	x	x	x	x	x	x	D-PS parameter
PARA_VIS	DigiVis	x	-	-	-	-	x	DigiVis parameter
PARA_EXP	D-PS	x	x	-	x	x	x	Nested FB parameter
MP_EXP	D-PS	x	-	-	-	-	x	Nested FB message data

Legend

x Function available

- Function not available

(1) Read from DigiVis — Variable can be accessed from DigiVis.

(2) Write from DigiVis — Variable can be changed (WRITE) from DigiVis or through a gateway.

(3) Write from D-PS — Variable can be changed by the program on the process station.

(4) Write from DigiTool — Variable can be changed (WRITE) from DigiTool in commissioning mode.

(5) Correct from DigiTool — Variable can be “corrected” from DigiTool in commissioning mode.

(6) Conf. in parameter mask — Variable can be changed from DigiTool in configuration mode (parameter mask).

Initial value
Value that the variable takes on each time the user function block instance is loaded.

Min. value / Max. value
The min. and max. values limit the range of values of a variable of storage type PARA_DPS. Staying within these limits is a plausibility criterion which is checked during configuration. If the limits are not upheld, the user function block instance will not be deemed plausible.

Ref-parameter
Reference to a value of a nested function block. The nested function block must have a name if it is to be referenced.

Comment
Comment relating to the variable for documentation purposes. The comment can have a maximum of 32 characters.

Predefined variables
The following predefined variables are for display of general function block data in the faceplate. Every user function block class has these variables available and they are not modifiable from within the class.

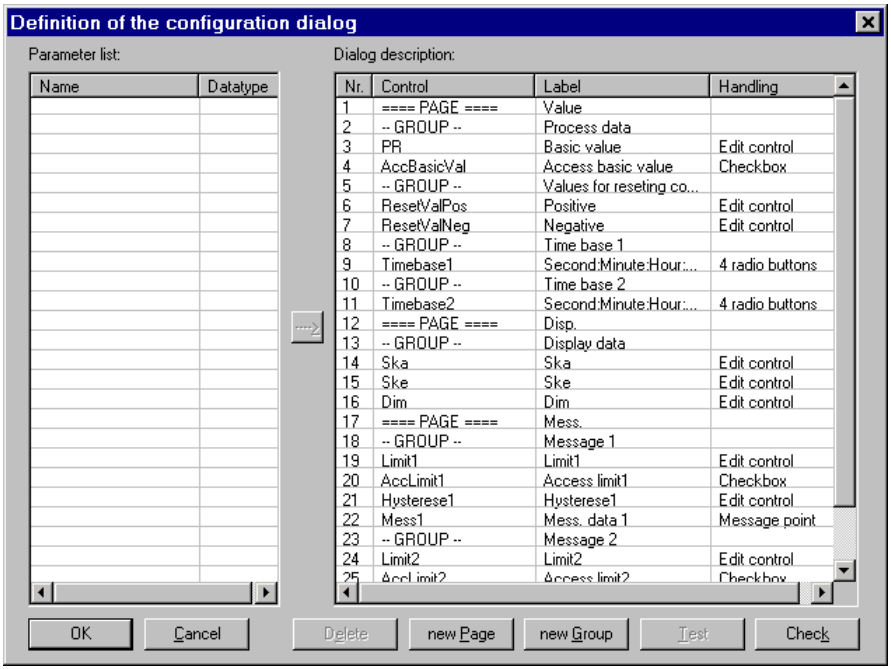
Name	Data type	Storage type	Comment
ClassName	TEXT	PARA_VIS	Contains the name of the user function block class.
TagName	TEXT	PARA_VIS	Contains the tag name of the user function block instance
ShortText	TEXT	PARA_VIS	Contains the short text of the user function block instance
LongText	TEXT	PARA_VIS	Contains the long text of the user function block instance
SelStat	BOOL	VAR_VIS	Indicates whether the faceplate is selected. TRUE = Faceplate is selected FALSE = Faceplate is not selected

H 2.1.2 Dialog editor

 → *Edit* → *Dialog editor*

Every user function block class has a default parameter mask. With this default parameter mask the tag name and the short and long text can be configured.
The dialog editor can be used to create a customized parameter mask for a user function block class. This parameter mask can then be used to assign other instance-specific parameter values.
The elements for use in the parameter mask are the parameters and messages of the user function block class. All variables available to the user function block class are shown in the left portion.
The dialog editor makes it possible to specify a text for display and input control (handling) setting for each parameter. In addition, the parameter dialog can be spread across multiple pages and the masks subdivided into group areas.

 If the default mask is extended, it must be given a new page.



th003us.bmp

Parameter list
List of parameters available for use in the parameter dialog of the user function block class

Name	Parameter name in interface editor (under user FB variables).
Data type	Data type of the parameter

Dialog description
Area for the definition of the parameter dialog for this user function block class. Every parameter dialog must begin with PAGE.

Control	Structuring element or parameter name	
	PAGE	introduces specifications for a new page in the parameter dialog
	GROUP/LINE	introduces specifications for a new group in the parameter dialog
Label	Text with which the structuring element or parameter will be displayed in the parameter dialog.	
Handling	Input control which will govern how parameter data is entered. May be specified for parameters only.	
OK	Close dialog editor and save changes.	

CANCEL Close dialog editor and discard any changes.

DELETE Delete the selected dialog entry.
A variable that has been assigned will continue to appear in the parameter list after deletion.

NEW PAGE A new dialog page is created.

☞ A dialog page must be given a name.

NEW GROUP A new dialog group is created.

TEST Switch the dialog editor into test mode.

☞ The dialog editor can only be switched into test mode when the dialog has passed the plausibility check.

CHECK Check the dialog for plausibility.

Procedure to create a parameter dialog:

1. Create new page
2. Copy all parameters requiring configuration; if necessary create additional page.
3. Enter texts for display.
4. If required, correct the input control (handling) settings.
5. If required, structure the parameter mask by dividing the parameters into groups.
6. Run plausibility check
7. Test the parameter dialog

Variables that are to participate in the parameter dialog must be copied from the parameter list to the parameter description.

→ mark parameters individually or in a block → '-->'

Every message or parameter line in the dialog editor corresponds to a line in the actual dialog. If the maximum number of lines possible in a dialog is exceeded, an error will occur during the plausibility check.
Each parameter is assigned input control (handling). A parameter's handling determines its appearance in the parameter dialog.
The possible input control settings depend on the data and storage type of the variable.

Input control	Data type	Example
Edit control	all Data types except BOOL	
Check box	BOOL	
<n> radio buttons	all integer data types	
Message point	Message (SYSTEM)	

Edit control
Edit control permits entering values for parameters of any data type.
In the actual dialog, the edit control field will have a fixed length. Inputs are scrollable. For exported parameters, the permissible input length is inherited from the nested function block.

Check box
Check boxes are used to specify the state of a parameter with data type BOOL.

<n> radio buttons
Radio button fields are used to specify discrete states. They can only be used with data types INT, UINT, DINT and UDINT. A min. and max. value must be specified for the parameter. With a field of <n> radio buttons, n texts for display must be entered in the form <text1>;<text2>;...;<textn>

Message line
A message line consists of the components of a message point, analogous to the standardized function blocks.
For messages with adjustable set point, the selection list for a set point type **(Type)** will be displayed, otherwise that field will be missing.

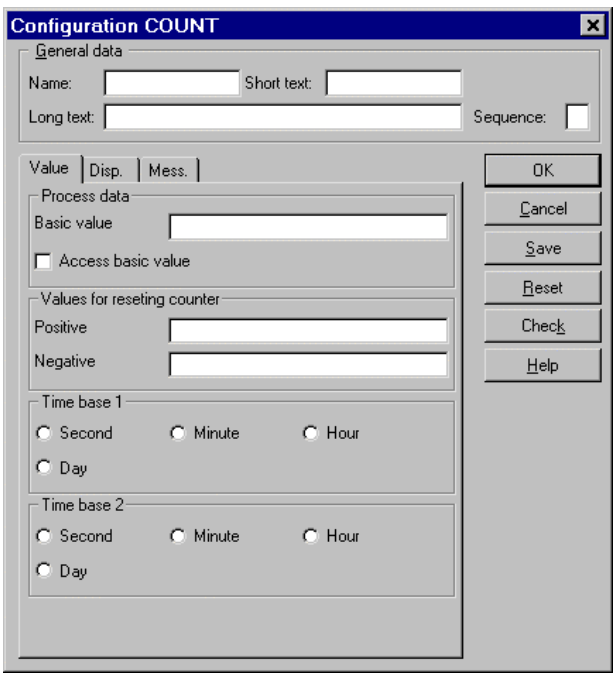
The standard buttons used in the standardized function blocks appear in the button area:

OK	Close parameter dialog and save values.
SAVE	Save values from the dialog.
CANCEL	Close parameter dialog without saving.
RESET	Restore the saved values to the parameter mask.
CHECK	Check the dialog for plausibility.
HELP	Call up help text for the parameter dialog.

Test
In test mode, the functioning of the parameter dialog created can be tested. The plausibility check is not available from this mode.

☞ Test mode is only available with parameter dialogs which have passed the plausibility check.

UFB parameter dialog, example



th004us.bmp

H 2.1.3 Text list

Texts for display are required both in the configuration of a user function block instance in DigiTool and in the graphic displays for operation and logging in DigiVis.
All texts for a user function block are referenced internally by their text ID's. The text contents are stored in the text list.

A new text can be defined during creation of a parameter dialog or a text object in the faceplate editor. In both cases, a text which has been previously specified for this function block can be selected from the list which is called up with the F2 key. If a new text is entered, a new text ID will be assigned, even if the text is identical with a text already in the list.

In order to facilitate translation outside of DigiTool, user function block texts can be exported and imported. Exported texts can be edited with any text editor.
Importing or editing a text list does not affect the plausibility of either user function block classes or instances.

Export text list

→ *Edit* → *Export text list*

The text list of the user function block will be written to a data carrier (e.g. hard disk, floppy disk) in ASCII format. A window appears in which the path and filename must be entered. This file can later be imported to other user function blocks—in the same or other projects—with *Import text list*
The file can be processed by other programs (e.g. word processors). The individual texts are arranged in lines with the following format:

< text ID >;< text >;< faceplate references >;< dialog references >

Example:

```
1;MAN;1;0
2;AUTO;1;0
3;Operating mode:;0;1
4;"MAN;AUTO";0;1
5;Extern;0;0
```

The exported file is a Unicode file.

Import text list

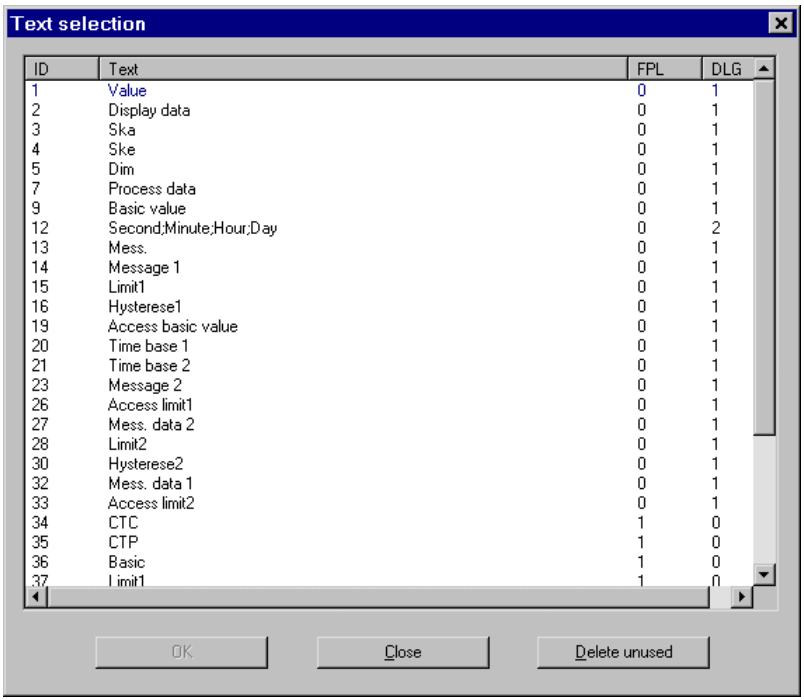
→ *Edit* → *Import text list*

A file that has been previously saved can also be read in from a data carrier (e.g. hard disk, floppy). A window appears in which the path and filename must be entered. Imported texts will be integrated into the text list; if an imported text has the same text ID as one already in the list, the text in the list will be replaced by the imported text. If any line in the file being imported deviates from the format described above, the import operation will be broken off at that point and none of the texts following the bad line will be imported.

Show text list

→ *Edit* → *Show text list*

The user function block text list is displayed in a separate window.



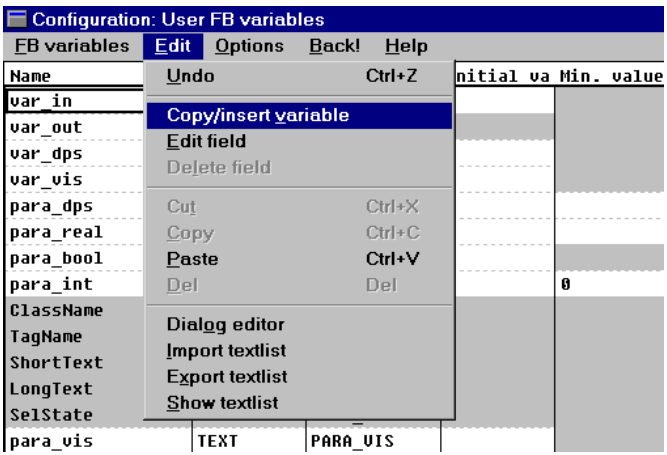
th005us.bmp

ID	Text ID of the text list entry
Text	Text entry
FPL	Number of faceplate editor references
DLG	Number of dialog editor references

CLOSE	Close the text list window
DELETE UNUSED	Delete all text entries which are not used (FPL and DLG references both equal to 0).

New text entries cannot be made in the text list here. The references are assigned by DigiTool.

H 2.1.4 Editing interface



di0285uk.bmp

Undo

→ *Edit* → *Undo*

The last change is canceled and the text is shown as it was before the last change. If the last change cannot be undone, the *Undo* menu item will not be selectable (reverse highlight).

Copy/insert variable

→ *Edit* → *Copy/insert variable*

Depending on the cursor position, either a new variable will be inserted (cursor on an empty name field) or an existing variable copied (cursor on an existing variable name).
For an empty field, a new variable name must be entered.
For a copy operation, a dialog is displayed with the old and new variable names. The new name field is initially filled in with the old name and must be changed.

Edit field

→ Select desired field with double click (highlight box)
The cursor appears on the last item of the entry
→ Click on desired item of entry in the field
→ Enter changes

The contents of the selected field may be changed. After the change has been made, a new window may appear, requesting confirmation and asking whether the change is to apply throughout the project or just in specific programs.

Delete field


→ Select desired field (highlight box, cursor appears on the last item of the entry)
→ *Edit* → *Delete field*


Entries in some specific fields cannot be deleted with this command.
Those fields are the name and type fields in the user FB variable list.

A variable may be deleted by selecting an entire line in the list.

A list entry may be deleted directly using the mouse and the DEL key as follows: click on the field to move the cursor into it, then move the cursor to the beginning of the section to be deleted; mark the section to be deleted by dragging the cursor over the text with the left mouse button depressed. Finally, press the DEL key to delete the marked text.

Cut


→ Select block → *Edit* → *Cut*

The selected block is removed from the list and saved in a buffer.
The block in the buffer can then be reinserted at any point using the *Paste* command.

Copy


→ Select block → *Edit* → *Copy*

The selected block is copied and saved in a buffer.
This block can then be reinserted at any point using the *Paste* command.

Paste


→ Select block → *Edit* → *Paste*

A block which has been saved in the buffer by *Copy* or *Cut* is inserted at the cursor position.


If variable names have been changed appropriately, the same window appears as with the menu item *Insert new variable.*

Delete


→ Select block → *Edit* → *Delete*

The selected block is deleted from the list.

Check

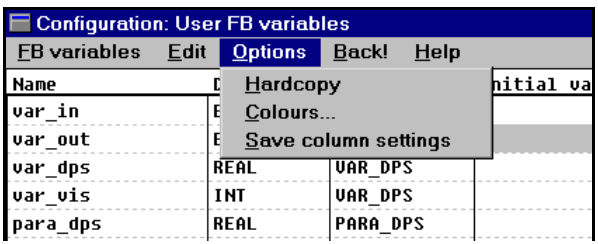
→ *FB variables* → *Check*

The interface editor data—i.e. the user function block variables—are checked for plausibility

Exit

→ *FB variables* → *Exit*

Back to previous screen.



di0293uk.bmp

Hardcopy

→ *Options* → *Hardcopy*

The screen contents are output to a printer.

Set colors

→ *Options* → *Colors*

The colors for unused variables may also be freely defined in the variable list.

Save column width

→ *Options* → *Save column width*

The column width setting is saved.

Back

→ *Back*

Back to previous screen.

H 2.2 Definition of user function block program

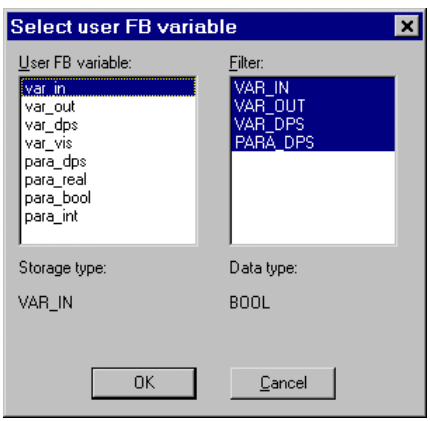
H 2.2.1 User function block program

Virtually all standardized function blocks and all functions are available in the configuration of a user function blocks. The positioning, parameterization, drawing of connecting lines, shifting and plausibility checking are done in the same way as for standardized function blocks.
It is not necessary to assign names to nested function blocks. If a name is given them at the user function block class level, it will be ignored by the user function block instances.

After choosing a variable field and pressing the F2 key, a window with the **user FB variables** list appears. The desired variable is then selected from this list. It is also possible to make entries directly in the input or output fields, but the variable entered must exist in the **user FB variables** list. New variables can only be entered under *System* → *User FB variables* .

☞ Process display variables (@) and exportable variables (#) may not be selected.

Variable selection



di0287uk.bmp

User FB variable	List of all variables which have been defined by the user for this function block. Depending on the filtering in effect, all variables will be displayed or just selected ones.
Filter	Only the selected storage types are displayed in the **User FB variable** window.
Storage type	Shows the storage types of the selected user-defined variables.
Data type	Shows the data types of the selected variables.

There are some limitations regarding the standardized function blocks that can be used. Particular exceptions are function blocks having an equivalent in DigiVis (such as trend acquisition blocks) or accessing special hardware (such as the Modbus interface block).

Existing user function blocks that have passed the plausibility check can be used in other user function blocks. A maximum of 8 levels of nesting is allowed. Recursive calling is not supported for user function blocks.

H 2.2.2 Messages

Message generation by user function blocks is accomplished by using nested function blocks. Any standardized or user function block with messages can be used.
The degree to which the message type can be changed, depends on the nested function block. The message type of all messages that refer to limit values can be changed in the user function block. All other message types are determined by the nested block and cannot be changed at the user function block level.
A message point comprises the following components:

- Message type (limit value type), see also **Engineering Reference Manual, Functions and Function Blocks, Abbreviations**
- Message priority
- Message text
- Hint text
- Display assignment
- Wave file

If the message point of a nested block is referenced, all associated components are automatically exported.

It is possible to configure “hidden” message points by configuring a message point in a nested block without referencing it in the interface editor. If a display is assigned to this point, a plausibility error will result.

H 2.3 Definition of user function block faceplate

H 2.3.1 General of faceplate editor

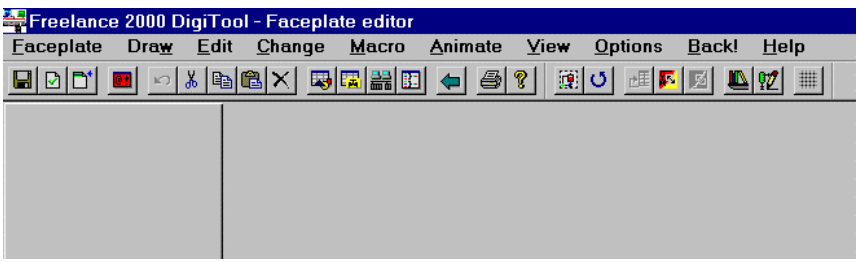
When a faceplate for user function block is selected in the project tree, the graphic editor is started in faceplate mode (faceplate editor).

→ **Project tree** → Double click on user function block faceplate

or

→ **Project tree** → Select user function block faceplate,
→ *Edit* → *Program*

Creation of the faceplate graphic is the same as the creation of a graphic display. See **Engineering Manual, Operator Station, Graphic display**.
Default static images are available for the overview faceplate. These images cannot be edited.



th013us.bmp

In principle, there is no difference between the graphic creation of a faceplate and that of a graphic display. The complete interface—menus, dialog masks, hint texts, error messages—is virtually the same as that of the graphic editor. (See **page H-28, Extensions in the faceplate editor**.)

It isn't possible to define a new variable in the faceplate editor.

The message points for the user function block are specified in the interface editor. In the faceplate editor, only message points of the user function block itself can be used; message points local to nested function blocks cannot be used.

Macros are handled in basically the same way in the faceplate editor as in the graphic editor.

H 2.3.2 Extensions in the faceplate editor

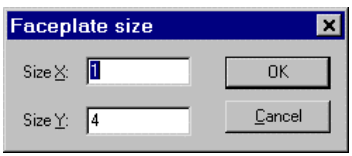
Faceplate size

A faceplate may have any rectangular size fitting within the 24-square process display format (*X-Size* up to 6 fields wide, by *Y-Size* up to 4 fields high). The desired faceplate size can be specified when the faceplate editor is called up for the first time for the creation of the faceplate, or with the *Specify size* menu item. A frame of the specified size will appear in the graphic display. The full graphic area will continue to be available for drawing.

If one or more graphic items is positioned wholly or partly outside of the prescribed frame, the frame will be displayed in a different color.

→ *Faceplate* → *Specify size*

→ *Faceplate* → *Optimize size*



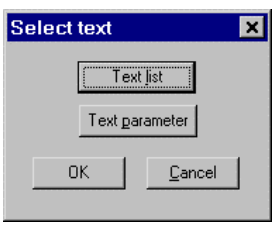
th014us.bmp

The prescribed faceplate size can be changed at any time with the *Specify size* menu item. The *Optimize size* menu item causes the system to set the faceplate size to the smallest possible value.

Display texts

With the graphic element *text* strings from the text list and the content of text variables for DigiVis (storage type: PARA_VIS; data type: TEXT) can be displayed. New static texts will be added automatically to the text list.

→ *Draw* → *Text* → F2



th043us.bmp

TEXT LIST — A text from the text list can be selected.

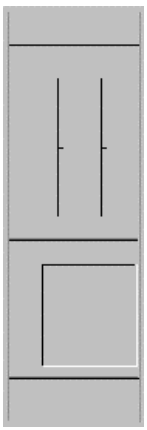
TEXT PARAMETER — A variable can be selected. The name of the variable will be shown in the faceplate editor.
The configured text for this variable will be shown in the faceplate in DigiVis.

☞ The content of a text parameter in the faceplate can only be changed by a *load* operation in commissioning mode in DigiTool.

H 2.3.3 Macros for faceplates

A macro library (ufp_sym1.bol) is available for use in creating faceplates. It contains faceplate elements commonly used in standardized function blocks.

Faceplate frame

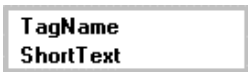


This macro contains a faceplate frame with dimensions X=1 and Y=4. Positions for the faceplate, graphic display, value display and button field are marked in the macro.

Macro name: UFP_frame

No macro parameters

Faceplate header with message information

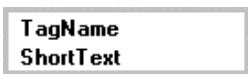


This macro contains a rectangle with two lines for text, one for the tag name and one for the short text. The color of the elements is determined by the message status and the *SelState* variable. The message used is singled out by the <most important message> function.

Macro name: Header_msg

Macro parameter	Comment
Message selection	Selection of messages from the user function block class, to be used for macro display color determination.

Faceplate header without message information

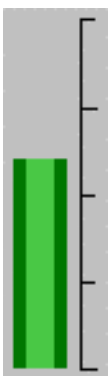


This macro contains a rectangle with two lines for text, one for the tag name and one for the short text. The color of the rectangle is determined by the value of the “SelState” variable.

Macro name: Header

No macro parameters

Set-point / current-value bargraph



This macro contains a set-point bargraph (dark green double bargraph) and a current-value bargraph (light green), with scale markings.

Macro name: Bargr_SP_PV

Macro parameter	Comment
Variable for PV barg.	Variable determining the height of the current-value bargraph
Variable for SP barg.	Variable determining the height of the set-point bargraph
Var. for scale start	Variable or constant for scale begin
Var. for scale end	Variable or constant for scale end

Output-value bargraph

This macro contains an output-value bargraph (yellow) with %-scale. The bargraph scale runs from 0% to100%.

Macro name: Bargr_OUT

Macro parameter	Comment
Variable for bargr.	Variable determining the height of the output-value bargraph

Limit bargraph

The macro contains a limit bargraph (arrow). The Y-extension is determined by the macro **Bargr_SP_PV**. The color of the arrow is determined by the status of a message. The message used is singled out by the <most important message> function.

Macro name: Bargr_limit

Macro parameter	Comment
Variable for limit	Variable determining the Y-position for the limit value
Var for scale start	Variable or constant for the scale begin
Var for scale end	Variable or constant for scale end
Message selection	Selection of messages from the user function block class for macro color display determination.

Split-point bargraphs

Macros are available for display of split point 1 and 2 respectively. The macros contain the split-point symbol with display of the characteristics type. The Y-position is determined by the split point. The Y-extension is determined by the macro **Bargr_OUT**. The split point display runs from 0% to 100%.

Macro name:	Bargr_Split1 Bargr_Split2

Macro parameter	**Comment**
Var. for split point	Variable determining the Y-position of the split point
Var. for up/down	Variable determining of the characteristic type FALSE = Rising characteristic TRUE = Falling characteristic

Message point symbol

These macros contain a site-specific message-type symbol. The <most important message> function is used for message display.
In the **MsgSym** macro, the color of the message-type symbol is determined by the status of the message. In the **MsgSym_red** macro, the message-type symbol is displayed in red.

Macro name:	MsgSym MsgSym_red

Macro parameter	**Comment**
Message selection	Choice of messages from the user function block class for macro display color determination.

Message type	**Comment**	**Display of the message type**
<no message type>	No message type configured	not visible
L	Limit type for underrun	
LL	Limit type for underrun	
LH	Message type for limit underrun/overrun	
HH	Limit type for overrun	
H	Limit type for overrun	

L_CE	Limit type for control difference underrun	
LL_CE	Limit type for control difference underrun	
LH_CE	Limit type for absolute value of control difference	
HH_CE	Limit type for control difference overrun	
H_CE	Limit type for control difference overrun	
LL_R, L_R, H_R, HH_R	Limit type for ratio limits	
DLh, DLm, DLs	Limit type for rate of change underrun	
DLLh, DLLm, DLLs	Limit type for rate of change underrun	
DHh, DHm, DHs	Limit type for rate of change overrun	
DHHh, DHHm, DHHs	Limit type for rate of change overrun	

Value display

Two macros are available for value display. In the **DispVal_s** macro, all text is displayed in 8p type size, while in the **DispVal_l** macro, all text is in 12p type size.
The macros contain a text and an alphanumeric display. The *_Standard* format is used for the alphanumeric display.

Macro name: DispVal_s
DispVal_l

Macro parameter	**Comment**
Variable to display	
Text to display	

Operable variable display

Two macros are available for display of operable variables. In the **DispVal_op_s** macro, all text is displayed in 8p type size, while in the **DispVal_op_l** macro, all text is in 12p type size.
The macros contain a text, an alphanumeric display and a rectangle for the operability function.

Macro name: DispVal_op_s
DispVal_op_l

Macro parameter	**Comment**
Variable to display	
Text to display	
Define operation	Operating action specification. The operability lock variable should be the same as that specified as the “Variable for lock” in the macro.
Variable for lock	Variable governing the display of the operability field: FALSE = grey background TRUE = blue background

Operating of operation mode

The macro contains two buttons for operating the operation mode. They are set up for displaying a Boolean state value (e.g. output **SMA**, C_ANA) and for operating an integer variable (e.g. parameter, C_ANA).

Macro name: Oper_Mode

Macro parameter	**Comment**
Text for Mode 0	Displayed for mode M0 (M0 button)
Text for Mode 1	Displayed for mode M1 (M1 button)
Variable to display	Boolean variable determining the operation mode displayed FALSE = Mode 0 active (M0 button) TRUE = Mode 1 active (M1 button)
Operation mode 0	Specification of the operating action associated with the M0 button. The “variable to display” should have the value FALSE as a result of the action.
Operation mode 1	Specification of the operating action associated with the M1 button. The “variable to display” should have the value TRUE as a result of the action.

H 2.4 Checking user function block classes

The plausibility check of a user function block class includes checks of the correctness of the interface declaration, the program, the dialogs and the faceplates. Only when no errors are present, is the user function block class declared plausible. The details of the testing are as follows:

- Invoking the plausibility check for the nested function blocks, using the parameter values specified.
- Checking the interface declaration (Do the referenced parameters and message points actually exist? Are the default value and value range consistent with the data type? Are the input control setting and the initial value consistent with the value range? Is there a name collision? ...)
- Faceplate plausibility checking

Because the error text of a nested standardized function block contains more information, that text is the one displayed in the error list in case of an error.

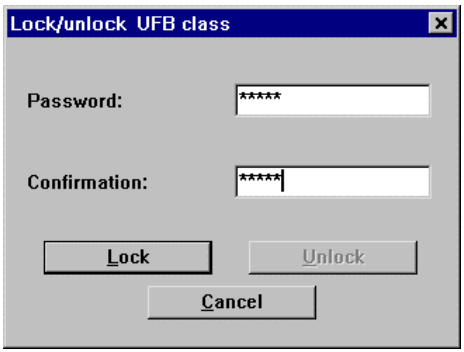
If a variable has been assigned min. and max. values, the maintenance of the variable within the range so defined will be checked as part of user function block instance plausibility checking.

☞ These limit values should be described for the user in the user FB help text.
See page **H-37**, **Help for user function blocks**.

H 2.5 Lock user function block class

It is possible to lock the implementation of a user function block class with a password.
Such locking makes it possible to hide the internal structure of the user function block (program, data structure) from the user, i.e. to make the user function block instances appear in their external representation only, like standardized function blocks. Similar to standardized function blocks, only the parameters are then configurable and commissionable.
A locked user function block cannot be modified

Select user function block class in project tree
→ *Options* → *Lock/Unlock UFB Class*



th015us.bmp

For the locking operation, the password must entered twice. To unlock the user function block, a single entry of the password is sufficient.

When a user function block class is locked, the following actions on the class are no longer possible:
- Calling up the corresponding program editor
- Calling up the faceplate editor
- Calling up the interface editor

For instances of a locked user function block class, only the parameter mask remains accessible, i.e. it is no longer possible to zoom in on the instance.

Locked user function block classes remain encrypted upon export.

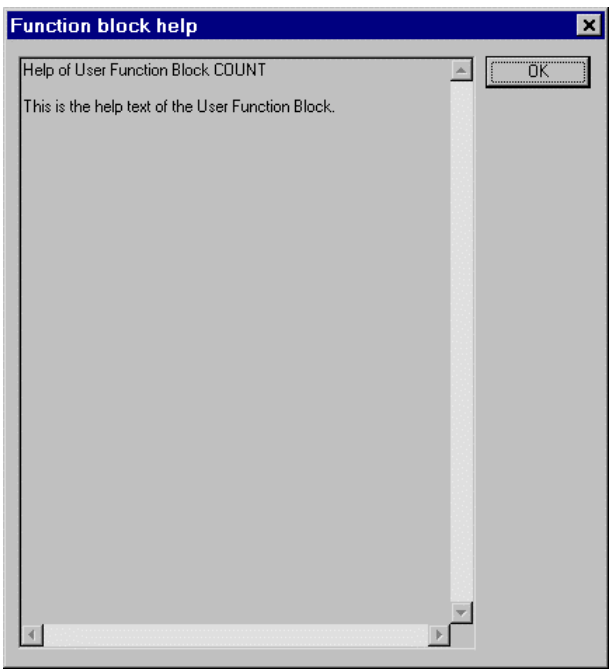
Messages from locked user function block instances are parameterized in the parameter dialog and reported under the tag name of the instance.

If a function block nested in a locked user function block has a tag name, messages from the nested block will also be displayed under the name of the nested block.

H 2.6 Help for user function blocks

→ Select user function block class in the project tree.
→ *Project* → *Comment*

The comment associated with the project tree junction of the user function block class is displayed as help text for the user function block instances. Any desired text can be entered or imported from an existing text for use as comment. The help text is called up via the HELP button in the user function block instance parameter dialog and displayed in a special window.



th012us.bmp

H 2.7 Export and import

A complete user function block class, or certain elements of it, can be exported or imported.
User function block classes from Freelance 2000 versions before V4 can still be used. The variables in user function blocks from earlier versions will be assigned the corresponding storage types as follows:

VAR	→	VAR_DPS
VAR_IN	→	VAR_IN
VAR_OUT	→	VAR_OUT

H 3 Commissioning

H 3.1 Load objects

All changes made to user function blocks proper are free of side effects. This means that when loading such changes, it is not necessary to halt either the resource or the task.

☞ User function block instances are loaded object by object to the process station. This is because, in contrast to standardized function blocks, they are made up of individual objects.

In the object list (*Show selected objects*), user function blocks with nested function blocks are thus displayed with more than one object under the same name. *Load* → *Changed objects* loads only those objects of a user function block which were changed.

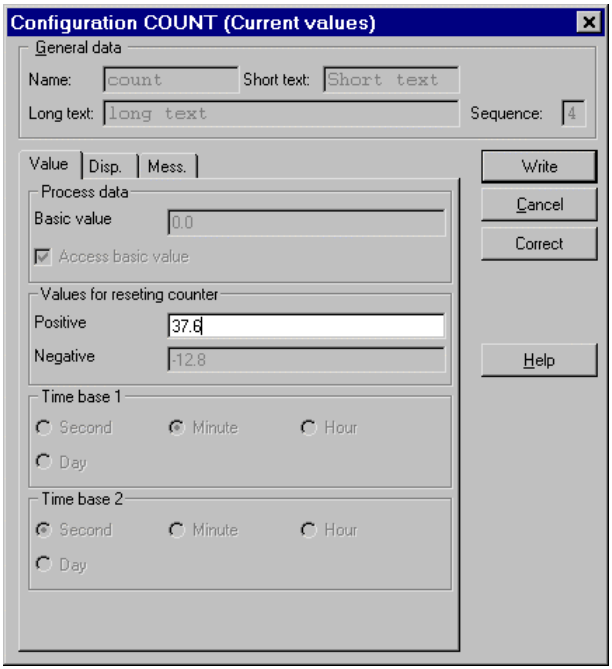
☞ A user function block class is loaded to DigiVis together with the project. This means that the DigiVis portion of the user function block must have been installed in the language of DigiVis on the DigiTool PC.

H 3.2 Read, write and correct

When reading, writing and correcting exported parameters (**PARA_EXP**), the action on the referenced parameters will be displayed for the nested function block. This display can under certain circumstances encompass many layers of nesting.

With an unlocked user function block instance, it is possible to zoom in on parameters of nested function blocks and write directly to them and to display their current values from the nested function block.

Parameters of nested function blocks (PARA_EXP) can be corrected. Correction will only be carried out when the plausibility check of the nested function block does not report any errors.
Correction operations on values in user function block instances work like those on standardized function block instances.
Variables with storage type PARA_VIS are not writable or correctable.



th006us.bmp

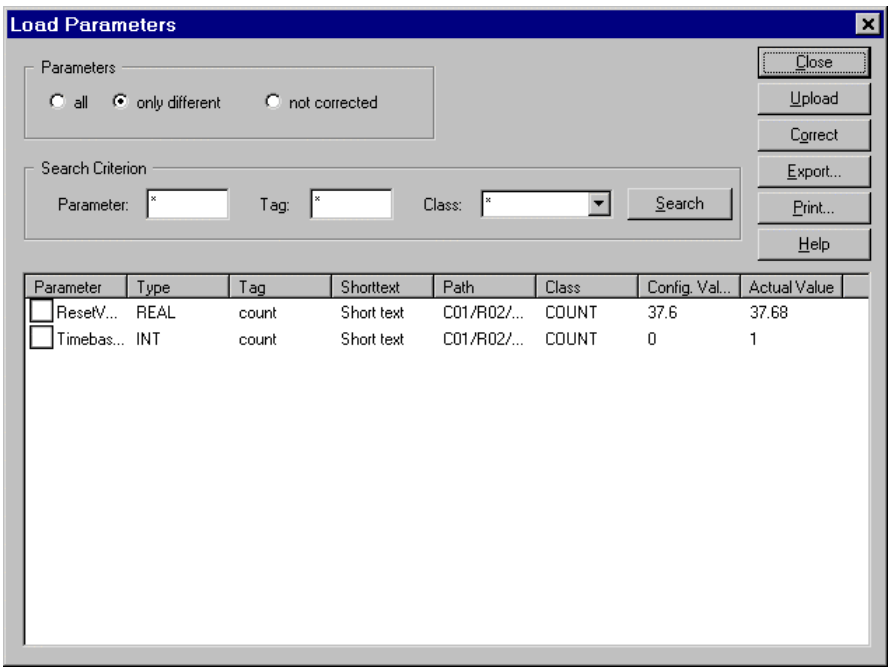
If variables of a user function block instance are being displayed in a value window or a trend window, these values will not be saved when commissioning mode is quit. When commissioning mode is reentered, the values previously in the value or trend window will no longer be available.

H 3.3 Load parameters

All variables having the PARA_DPS and PARA_EXP storage types are available for parameter upload.

☞ Every parameter appears only once in the parameter upload list.

If an exported parameter has a tag name as its source, then the parameter will only be accepted in the most outside point of use in the upload list. This will cause the parameter to be missing from all nested function blocks with tag names.

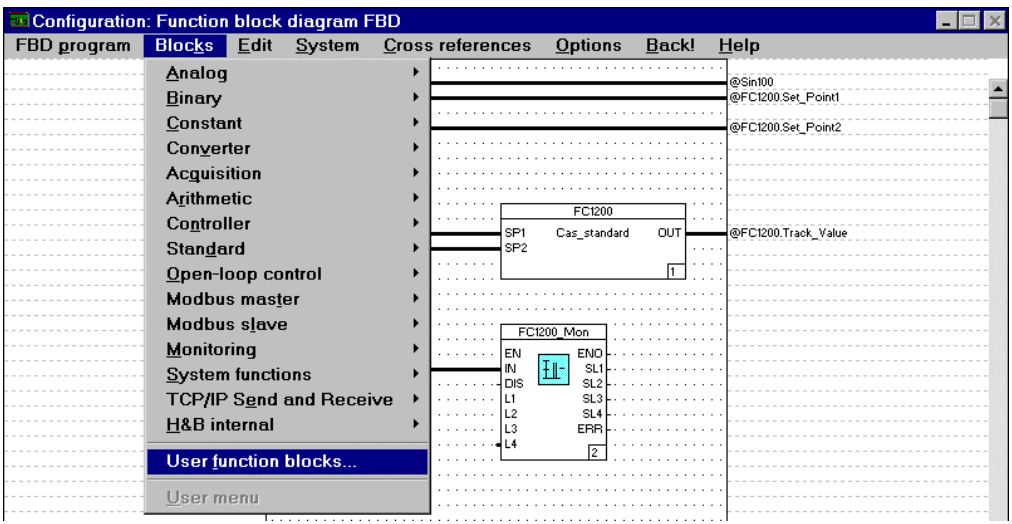


th007us.bmp

H 4 Generation of User Function Block Instances

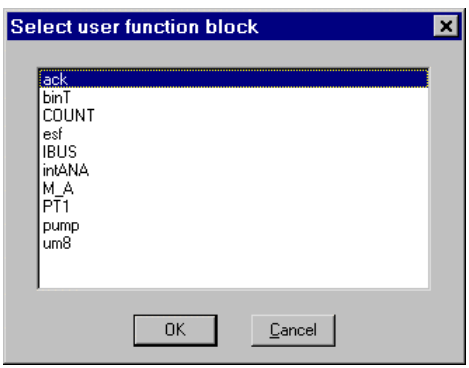
H 4.1 Create new user function block instance

A function block instance is created by first choosing a class out of a list of user function block classes. Instances can only be created from classes which have passed the plausibility check.



di0289uk.bmp

→ *Blocks* → *User function blocks ...*
All defined function blocks which have passed plausibility appear in the **Select user function block** window.
→ Choose and position the desired function block.



di0290uk.bmp

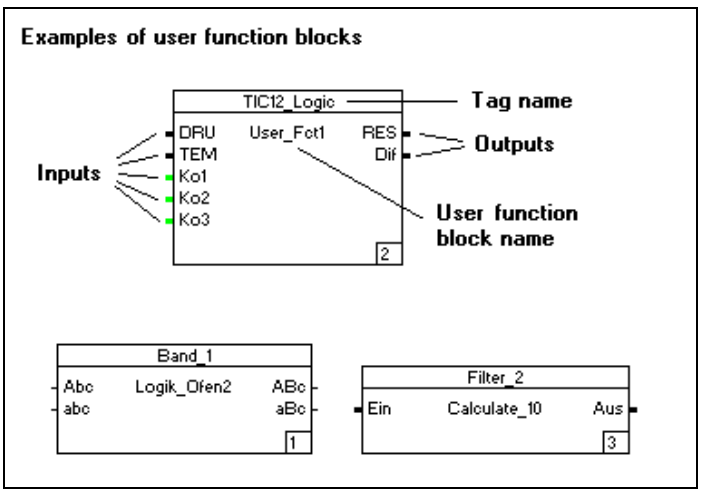
After the user function block has been selected from the list, it is positioned in the program and the variables can be connected.

Changes to the user-defined function block structure (connecting lines, addition of blocks or changes to blocks) can only be made in the **user function block class.** See page **H-48 Modification of User Function Blocks**.

H 4.2 Using user function blocks

H 4.2.1 Pin layout

FBD/LD program
The size of a user function block depends on the number inputs and outputs. The class name of the user function block appears in the middle of the symbol. The tag name allocated in the application appears in the upper portion of the symbol. This name also appears in the list of allocated tags, with the notation of the associated user function block. The inputs and outputs are labeled with their pin designations.

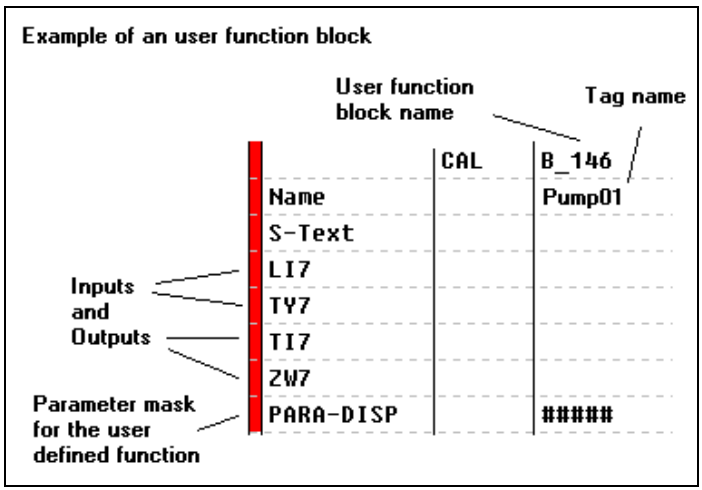


di0288uk.bmp

IL Program

The length of the user function block in the IL program depends on the number of inputs and outputs. The class name is entered behind the CAL call in the **Operand** column. The tag name given in this application is in the line directly below. This name also appears in the list of allocated tags with the notation of the user function block name.

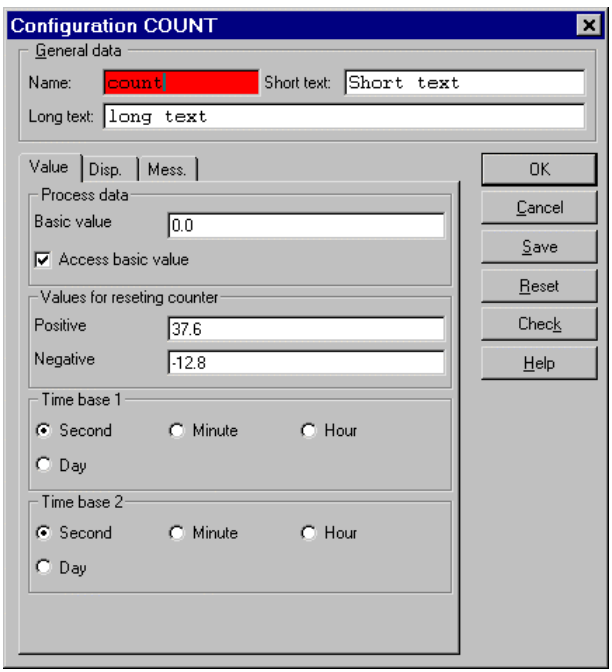
The first three letters of the user-defined function block input and output designations appear on the inputs and outputs.



di0277uk.bmp

H 4.2.2 Modification of parameter data

The parameter dialog is opened with a double click on the block. There, the allocated tag name of the function must be entered.
If an individualized parameter dialog was created for the user function block, the individual parameter values for the instance can be entered here.
Any additional pages of the parameter dialog can be brought up using the **tab control.**
Parameters and messages of a user function block instance are filled in with the default values from the class declaration the first time the parameter dialog of the instance is called up. They can be adjust as required for each instance.



di0291uk.bmp

General data

Name	The name may be up to 12 characters in length and must be unique within the project. Entry here is required.
Short text	Up to 12 characters, all characters are allowable.
Long text	Up to 30 chars., all chars. allowable.

OK — The parameter dialog is closed and the parameter values are **saved**.

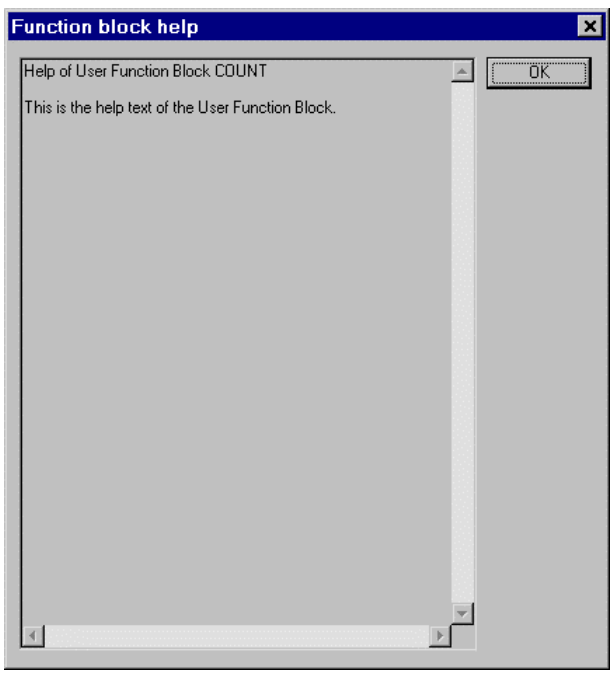
CANCEL — The parameter dialog is closed **without saving** the parameter values. A warning appears if parameter value changes are lost.

SAVE — The current parameter values are **saved** but the window remains open.

RESET — The values in the parameter window are reset completely to the **preset default values**. Any values previously saved and differing from the default settings can be retrieved by canceling and reopening the parameter window.

CHECK — The user function block instance is checked for plausibility with the current parameters, even if they have not been saved. All nested function blocks are also subjected to plausibility checking.

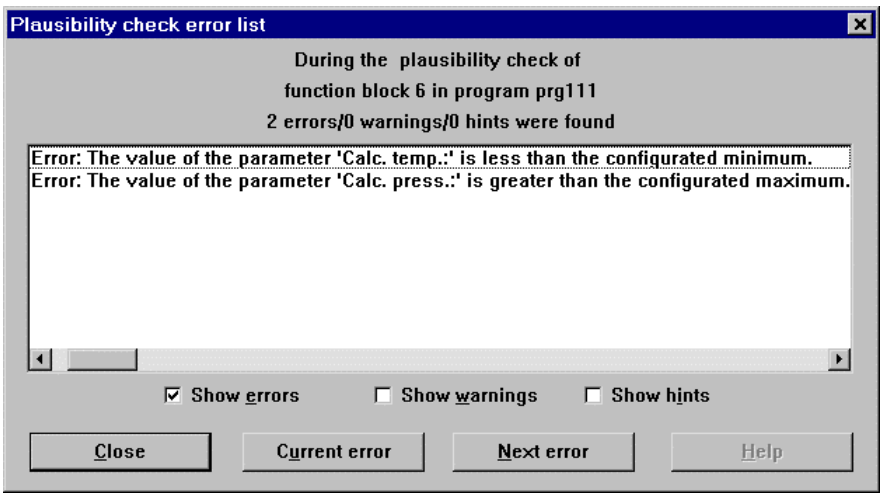
HELP — Help is provided for the user function block. The comment text associated with the user function block class is displayed as help text.



th012us.bmp

H 4.2.3 Check of instances

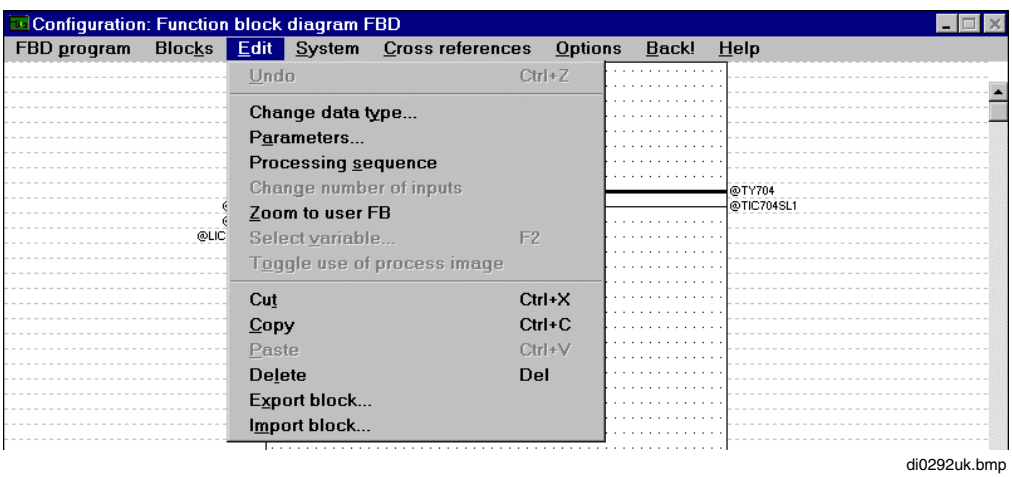
When parameters are entered for a user function block instance, they are checked against any value ranges previously entered in the interface editor. If any ranges are exceeded, the user function block instance is marked as implausible.



th016us.bmp

H 4.2.4 Zoom to user function block

Entries in the parameter masks of the nested function blocks can be made from their respective programs. Such entries are only possible with user function block instances which are not locked.



di0292uk.bmp

→ Select user-defined function block symbol with single cursor click
→ *Edit* → *Zoom to user FB*
the user function block program is displayed
→ Double click on the desired block
→ make desired changes in the parameter mask that appears

In the case of unlocked user function block instances, instance-specific values for variables with PARA_EXP storage type can be changed either in the user function block instance dialog or in the dialog of the nested block. In contrast to the situation in earlier versions, the nested blocks do not require allocated tag names.

H 4.3 Using user function block faceplates

The group display editor can handle any number of rectangles within the 24-square (6 fields wide by 4 fields high) faceplate format. Faceplates in a group display may not cover one another, even partially.
Default static images are available for the overview faceplate. These images cannot be edited.

H 5 Modification of User Function Blocks

Changes to user function block can only be made in the **user function block pool,** i.e. for the user function block class.
If inputs or outputs are added, the user function block must be reinstalled in programs where it is used. In such a case, the changed user function block, i.e. the instances, are marked in red in the programs.

☞ Only user function block classes which are not locked can be changed.

In general, user function block instances become implausible when their class becomes implausible. In addition, all user function blocks which refer to a user function block when the class referred to becomes implausible. Changing of comments has no effect on plausibility. Instances with no corresponding classes—which occur for example, when the class is deleted or moved to the pool—are displayed as incompatible (in red).

Changes to the user function block interface
After changes to the interface of a user function block, a plausibility check of the associated faceplate is required. If the change had an effect on the faceplate, then the faceplate must be loaded to DigiVis.

Changes to the UFB text list
After a change to the text list of a user function block, a plausibility check of the associated faceplate is required. If the change had an effect on the faceplate, then the faceplate must be loaded to DigiVis.

Changes made in commissioning
During interface definition for a user function block, the system makes sure that every component of the block is assigned to exactly one resource—process station or DigiVis. During commissioning, only process station components can be manipulated. It is thereby assured, that the faceplate is not affected by commissioning, and commissioning will never require loading to DigiVis.

Changes to the faceplate
When the faceplate is created, only previously defined components of the user function block can be accessed. Changes to the graphic cannot have any effect on the program portion of the function block.

Additional changes and effects relating to user function block classes and instances are as follows:

Effects of changes to user function block classes

Change	Effect	Comment
Rename a class	1. d.	A warning is issued before the class is renamed.
Delete a class	d.	No 1 or 2, because the class no longer exists.
Move class to class pool	d.	Same as deleting a class.
Move class to project pool and then back	1. b. 1. d.	If no substantial change was made. If a substantial change was made.
Delete class and then create or import it again	1. b. 1. d.	f no substantial change was made. If a substantial change was made.
Change class sequence (move to UFB pool)	1. b.	
Add variables with storage type VAR_IN or VAR_OUT	1. d.	
Delete variables with storage type VAR_IN or VAR_OUT	1. d.	
Add variables with storage type PARA_DPS or PARA_VIS	1. b.	
Delete variables with storage type PARA_DPS or PARA_VIS	1. c.	The configured value is lost.
Add variables with storage type PARA_EXP or MP_EXP	1. b.	
Delete variables with storage type PARA_EXP or MP_EXP	1. b.	☞ The configured value in a nested function block is not lost.
Add variables with storage type VAR_DPS or VAR_VIS	1. b.	
Delete variables with storage type VAR_DPS/VAR_VIS	1. b.	
Change the program structure	1. b.	
Add function block call in the program	1. b.	
Delete function block call in the program	1. c.	The data configured for the deleted program is lost.
Rename program junction in project tree	2. a.	
Delete program junction in project tree	1. d.	
Change or delete tag name of a nested function block	1. b.	
Faceplate changes	1. b.	
Text list changes	1. b.	
Parameter dialog changes	1. b.	

Effects of changes in the user function block interface

Change to	Effect				
	Data type	Initial value	Value range	Reference parameter	Comment
VAR_IN Var_OUT	1. d.	1. b.			2. a.
PARA_DPS	1. b.	1. b.	1. b. or 1. e. [1]		2. a.
PARA_VIS	1. b.	1. b.	1. b. or 1. e.		2. a.
PARA_EXP MP_EXP				1. b.	2. a.
VAR_DPS	1. b.	1. b.			2. a.
VAR_VIS	1. b.	1. b.			2. a.

If a class A function block is used by a class B function block, then, in general, the change status of A is passed up to B.

Legend:

Possible effects on a user function block class:

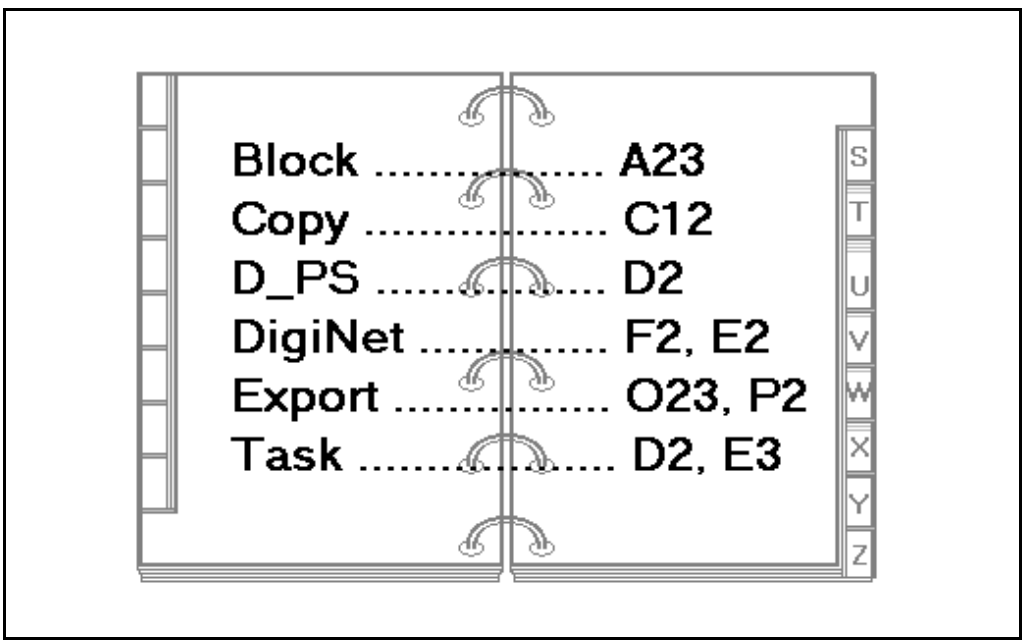
Abbreviation	Effect
1.	Class is made implausible
2.	Class remains plausible

Possible effects on a user function block instance

Abbr.	Instance-specific parameterization remains intact	Instance remains plausible	Instance becomes implausible (red)	Change required at the instance level
a.	yes	yes	no	no
b.	yes	no	no	no
c.	partially	no	no	no
d.	no	no	yes	no
e.	yes	no	no	yes

[1] The effect of the change depends on the validity of the initial value and the configured values in the new value range.

X Index


Block A23
Copy C12
D_PS D2
DigiNet F2, E2
Export O23, P2
Task D2, E3
S
T
U
V
W
X
Y
Z

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

Z

Gross Automation, 1725 South Johnson Road, New Berlin, WI 53146, www.ssacsales.com, 800-349-5827