PROCESS AUTOMATION

Freelance 2019

Engineering Manual
IEC 61131-3 Programming

PROCESS AUTOMATION

Freelance 2019

Engineering Manual
IEC 61131-3 Programming

Document Number: 3BDD012504-111
Revision: A
Release: January 2019

Notice

This document contains information about one or more ABB products and may include a description of or a reference to one or more standards that may be generally relevant to the ABB products. The presence of any such description of a standard or reference to a standard is not a representation that all of the ABB products referenced in this document support all of the features of the described or referenced standard. In order to determine the specific features supported by a particular ABB product, the reader should consult the product specifications for the particular ABB product.

ABB may have one or more patents or pending patent applications protecting the intellectual property in the ABB products described in this document.

The information in this document is subject to change without notice and should not be construed as a commitment by ABB. ABB assumes no responsibility for any errors that may appear in this document.

Products described or referenced in this document are designed to be connected, and to communicate information and data via a secure network. It is the sole responsibility of the system/product owner to provide and continuously ensure a secure connection between the product and the system network and/or any other networks that may be connected.

The system/product owners must establish and maintain appropriate measures, including, but not limited to, the installation of firewalls, application of authentication measures, encryption of data, installation of antivirus programs, and so on, to protect the system, its products and networks, against security breaches, unauthorized access, interference, intrusion, leakage, and/or theft of data or information.

ABB verifies the function of released products and updates. However system/product owners are ultimately responsible to ensure that any system update (including but not limited to code changes, configuration file changes, third-party software updates or patches, hardware change out, and so on) is compatible with the security measures implemented. The system/product owners must verify that the system and associated products function as expected in the environment they are deployed.

In no event shall ABB be liable for direct, indirect, special, incidental or consequential damages of any nature or kind arising from the use of this document, nor shall ABB be liable for incidental or consequential damages arising from use of any software or hardware described in this document.

This document and parts thereof must not be reproduced or copied without written permission from ABB, and the contents thereof must not be imparted to a third party nor used for any unauthorized purpose.

The software or hardware described in this document is furnished under a license and may be used, copied, or disclosed only in accordance with the terms of such license. This product meets the requirements specified in EMC Directive 2014/30/EU and in Low Voltage Directive 2014/35/EU.

Trademarks

All rights to copyrights, registered trademarks, and trademarks reside with their respective owners.

Table of contents

About this book

Use of warning, caution, information, and tip icons 17
Terminology 18
Document conventions 18

1 - Variables

1.1 General Description - Variables 21
1.2 Data types 22
1.2.1 Overview of simple data types 22
1.3 Variable list 23
1.3.1 Call the variable list 23
1.3.2 Structure of the variable list 23
1.3.3 Edit the variable list 26
1.3.4 Initial values 28
1.3.5 Normal view and station view 30
1.3.6 Close 31
1.4 Edit variable list entries 31
1.4.1 Undo 32
1.4.2 Create a new variable in the list 32
1.4.3 Create a new variable in a program 33
1.4.4 Insert an existing variable in a program 33
1.4.5 Edit a field in the list 34
1.4.6 Delete field 35
1.4.7 Delete unused variables 35
1.4.8 Delete I/O allocation 36
1.4.9 Edit block 36
1.4.10 Export 38

1.4.11 Import....39
1.4.12 Cross references....41
1.4.13 Station access....43
1.4.14 Assign block to resources automatically....43
1.4.15 Assign block to resources manually....44
1.4.16 Assign block to process image....45
1.5 Options....45
1.5.1 Print....45
1.5.2 Adjust colors....45
1.5.3 Save column settings....46
1.5.4 Auto accept....46
1.5.5 Save filter....46
1.5.6 Clear filter....46
1.5.7 Show saved filters....46
1.6 System variables....46
1.6.1 System variables with project information....47
1.6.2 System variables with resource information....48
1.6.3 System variables with information of a redundant resource....50
1.6.4 System variables for power fail on voltage failure....51
1.6.5 System variables for error handling task....52
1.6.6 System variables for I/O communication....52
1.6.7 System variables with information for lateral communication....53
1.7 Structured data types....54
1.7.1 Call definition of structured data types....54
1.7.2 Define a new data type....54
1.7.3 Create data type components....54
1.7.4 Insert a new variable with structured data type....55
1.7.5 Use a structured variable in a program....56

2 - Tags

2.1 General description - Tag list....59
2.1.1 Call the tag list....59
2.1.2 Structure of the tag list....60

2.1.3 Edit the tag list 62
2.1.4 Normal view and station view 65
2.1.5 Close 66
2.2 Edit tag list entries 67
2.2.1 Undo 67
2.2.2 Insert new tag in the list 68
2.2.3 Edit a field in the list 69
2.2.4 Delete field 69
2.2.5 Delete unused tags 69
2.2.6 Edit block 70
2.2.7 Export 72
2.2.8 Import 73
2.2.9 Cross references 75
2.2.10 Station access 76
2.2.11 Area 77
2.2.12 Change function block type 78
2.2.13 Access rights 78
2.2.14 User groups 79
2.3 Options 79
2.3.1 Print 79
2.3.2 Adjust colors 79
2.3.3 Save column settings 79
2.3.4 Auto Accept 79
2.3.5 Save filter 80
2.3.6 Clear filter 80
2.3.7 Show saved filters 80

3 - OPC items

3.1 General Description - OPC items 81
3.1.1 Call OPC item list and browse for OPC items 81
3.1.2 Structure of the OPC item list 82
3.1.3 Sort the OPC item list 83
3.1.4 Edit the OPC item list 84

3.2 Assign variable 86
3.3 Standard library of OPC_FB-Classes 94
3.3.1 OPC_FB-CLASS and instances 94
3.3.2 Create an OPC_FB-CLASS library 94
3.4 Definition of OPC_FB-CLASS 95
3.4.1 OPC_FB-CLASS interface 95
3.4.2 Modify an OPC_FB-Classes 98
3.4.3 Create an OPC_FB-CLASS 100
3.4.4 Faceplate for an OPC_FB-CLASS 101
3.4.5 Check OPC_FB-CLASS 102
3.4.6 Lock OPC_FB-CLASS 102
3.4.7 OPC_FB-CLASS comments 103
3.4.8 Export / Import 103
3.5 Tag instantiation 103
3.5.1 Instantiate All 105

4 - Libraries

4.1 Library – User interface 107
4.1.1 Specify own library list 108
4.1.2 Specify favorites list 110
4.1.3 All blocks 111
4.1.4 User function blocks 113
4.1.5 Sort elements in the list 113
4.1.6 Insert library elements into a program 114
4.1.7 Hide and show the Library explorer 115

5 - Function Block Diagram (FBD)

5.1 General Description - Function Block Diagram 117
5.1.1 Create an FBD program 118
5.1.2 Copy an FBD program 118
5.1.3 Delete an FBD program 119
5.1.4 Call the FBD program editor 119
5.1.5 Close FBD program 119

5.2 Representation of the Function Block Diagram 120
5.2.1 User interface of the FBD editor 120
5.2.2 Modify default settings 121
5.2.3 Display program information 123
5.3 Description of FBD program elements 124
5.3.1 Connections and Lines 124
5.3.2 Variables and Constants 125
5.3.3 Blocks 126
5.3.4 Comment fields 127
5.4 Parameterize FBD program elements 128
5.4.1 Parameter definition of function blocks 128
5.4.2 Parameterize comment fields 132
5.4.3 Change the processing sequence of the blocks 132
5.4.4 Define favorites list 133
5.5 Edit an FBD Program 133
5.5.1 Draw signal flow lines 133
5.5.2 Insert FBD elements 138
5.5.3 Change number of inputs 141
5.5.4 Display and change data types 142
5.5.5 Invert a block terminal 143
5.5.6 Change variables 143
5.5.7 Cross references 144
5.5.8 Insert or delete columns and rows 145
5.5.9 Block operations 146
5.5.10 Undo an action 151
5.5.11 Program administration functions 151
5.6 Commissioning the Function block diagram (FBD) 153

6 - Instruction List (IL)

6.1 General Description – Instruction List 157
6.1.1 Create an IL program 158
6.1.2 Copy an IL program 158
6.1.3 Delete an IL program 159

6.1.4 Call the IL program editor 159
6.1.5 Close IL program 159
6.2 Representation of the Instruction List 160
6.2.1 User interface of IL editor 160
6.2.2 Modify default settings 162
6.2.3 Display program information 163
6.2.4 Define favorites list 164
6.3 Edit an IL Program 164
6.3.1 Acceptable data types for IL operators and functions 165
6.3.2 Call IL operators 168
6.3.3 Insert function blocks into an IL program 178
6.3.4 Cross references 181
6.3.5 Program administration functions 182
6.4 Commissioning the Instruction list (IL) 185

7 - Ladder Diagram (LD)

7.1 General Description – Ladder Diagram 187
7.1.1 Rules for processing a Ladder Diagram program 188
7.1.2 Create an LD program 189
7.1.3 Copy an LD program 190
7.1.4 Delete an LD program 190
7.1.5 Call the LD program editor 190
7.1.6 Close LD program 191
7.2 Representation of the Ladder Diagram 192
7.2.1 User interface of the LD editor 192
7.2.2 Modify default settings 193
7.2.3 Display program information 195
7.2.4 Define favorites list 196
7.3 Description of the Ladder Diagram elements 196
7.3.1 Connections and lines 196
7.3.2 Contacts 198
7.3.3 Coils 199
7.3.4 Variables and constants 201

7.3.5 Function blocks 203
7.3.6 Jumps and returns 204
7.3.7 Labels 206
7.4 Parameterize Ladder Diagram elements 206
7.4.1 Parameterize a contact 207
7.4.2 Parameterize a coil 208
7.4.3 Parameterize a variable 209
7.4.4 Parameterize a jump 209
7.4.5 Parameterize a label 210
7.4.6 Parameterize function blocks 210
7.5 Edit an LD program 210
7.5.1 Representation of the signal flow lines 210
7.5.2 Draw lines 211
7.5.3 Insert LD elements and function blocks 214
7.5.4 Insert or delete columns and rows 215
7.5.5 Cross references 216
7.5.6 Block operations 218
7.5.7 Program administration functions 222
7.6 Commissioning the Ladder diagram (LD) 224

8 - Structured Text (ST)

8.1 General Description – Structured Text 227
8.1.1 Create an ST program 228
8.1.2 Copy an ST program 228
8.1.3 Delete an ST program 229
8.1.4 Call the ST program editor 229
8.1.5 Close ST program 229
8.2 Representation of the Structured Text 229
8.2.1 User interface of the ST editor 229
8.2.2 Syntax coloring 231
8.2.3 Modify default settings 232
8.2.4 Display program information 233
8.2.5 Define favorites list 234

8.3 Description of the ST program elements 234
8.3.1 Language elements 234
8.3.2 Types 239
8.3.3 Variables and function blocks 241
8.3.4 Expressions 245
8.3.5 Statements 247
8.3.6 Limits of the system 257
8.3.7 Examples 259
8.4 Edit an ST program 264
8.4.1 Insert ST elements 264
8.4.2 Insert variables and function blocks 265
8.4.3 Working with variables 269
8.4.4 Working with functions 270
8.4.5 Working with function blocks 272
8.4.6 Program user-defined function blocks 277
8.5 General processing functions 279
8.5.1 Bookmarks 279
8.5.2 Breakpoints 280
8.5.3 Find and replace 281
8.5.4 Goto line 282
8.5.5 Block operations 283
8.5.6 Cross references 287
8.5.7 Program administration functions 288
8.6 Commissioning structured text 291
8.6.1 User interface for commissioning 291
8.6.2 Display of online data 292
8.6.3 Error tracing 292

9 - Sequential Function Chart (SFC)

9.1 General Description – Sequential Function Chart 293
9.1.1 Create an SFC program 295
9.1.2 Call SFC program editor 295
9.1.3 Close SFC program 295

9.1.4 Basic rules ... 296
9.1.5 Example of how to edit ... 296
9.2 Structure of the Sequential Function Chart ... 298
9.2.1 SFC program user interface ... 298
9.2.2 Display program information ... 299
9.2.3 Drawing help ... 300
9.3 Edit SFC Elements ... 301
9.3.1 Initial step ... 302
9.3.2 Step ... 302
9.3.3 Jump ... 303
9.3.4 Transition ... 303
9.3.5 Vertical line ... 304
9.3.6 Horizontal sequence selection line ... 304
9.3.7 Sequence selection divergence start ... 305
9.3.8 Sequence selection divergence add ... 305
9.3.9 Sequence selection convergence add ... 306
9.3.10 Sequence selection convergence end ... 306
9.3.11 Horizontal simultaneous sequence line ... 306
9.3.12 Simultaneous sequence divergence start ... 307
9.3.13 Simultaneous sequence divergence add ... 307
9.3.14 Simultaneous sequence convergence end ... 307
9.3.15 Simultaneous sequence convergence add ... 308
9.4 Edit SFC structure ... 309
9.4.1 Shift blocks ... 310
9.4.2 Undo ... 310
9.4.3 Edit columns / lines ... 311
9.4.4 Parameterize SFC program elements ... 315
9.4.5 Edit program ... 321
9.4.6 Define criteria window ... 321
9.4.7 Define display access ... 329
9.4.8 Parameterize SFC program ... 330
9.4.9 Edit elements ... 335

9.4.10 Export and import blocks....337
9.4.11 Program administration functions....338
9.5 Commissioning the SFC program....340
9.5.1 Operation dialog SFC program....342
9.5.2 Step operating dialog....344
9.5.3 Transition operation dialog....345
9.5.4 Step states....346
9.5.5 Step action execution....347
9.5.6 Display of steps in the SFC program....347
9.5.7 Transition states....348
9.5.8 Display of transitions in the SFC program....349

10 - User Function Blocks

10.1 General Description – User Function Blocks....351
10.1.1 User function block - classes and instances....353
10.1.2 Create user function block pool....354
10.1.3 Create a user function block class....354
10.1.4 Create a user function block program....355
10.1.5 Create a user function block faceplate....355
10.2 Definition of User Function Block Classes....356
10.2.1 Interface of a user function block....356
10.2.2 Edit interface of a user-defined function block....363
10.2.3 Parameter dialog of a user function block....366
10.2.4 Text list....371
10.2.5 User function block program....374
10.2.6 User function block faceplate....376
10.2.7 Check user function block classes....379
10.2.8 Lock user function block class....380
10.2.9 Help for user function blocks....381
10.2.10 Export and import....382
10.3 Commissioning....383
10.3.1 Load objects....383
10.3.2 Read, write and correct....383

10.3.3 Load parameters 384
10.4 Generate instances of user function blocks 386
10.4.1 Create new user function block instance 386
10.4.2 Using user function blocks 387
10.4.3 Use faceplates of user function blocks 393
10.5 Modification of user function blocks 393

11 - Debugger

11.1 General description – Debugger 399
11.1.1 Fault tracing with the debugger 399
11.1.2 Breakpoints 400
11.2 Debugger interface 402
11.2.1 Breakpoint list 402
11.2.2 Watch window 404
11.3 Working with the debugger 407
11.3.1 Starting the debugger 407
11.3.2 Edit breakpoint 408
11.3.3 Task state 409
11.3.4 Single step 411
11.3.5 Watch values 412
11.3.6 Go 412
11.3.7 Stop debugger 412
11.3.8 Typical examples of errors 413
11.4 Breakpoint functions 418
11.4.1 Mark breakpoints 418
11.4.2 Event log 420

Index

About this book

Use of warning, caution, information, and tip icons

This publication includes **Warning**, **Caution**, and **Information** where appropriate to point out safety related or other important information. It also includes **Tip** to point out useful hints to the reader. The corresponding symbols should be interpreted as follows:

Electrical warning icon indicates the presence of a hazard which could result in *electrical shock.*

Warning icon indicates the presence of a hazard which could result in *personal injury.*

Caution icon indicates important information or warning related to the concept discussed in the text. It might indicate the presence of a hazard which could result in *corruption of software or damage to equipment/property.*

Information icon alerts the reader to pertinent facts and conditions.

Tip icon indicates advice on, for example, how to design your project or how to use a certain function

Although **Warning** hazards are related to personal injury, and **Caution** hazards are associated with equipment or property damage, it should be understood that operation of damaged equipment could, under certain operational conditions, result in degraded process performance leading to personal injury or death. Therefore, comply fully with all **Warning** and **Caution** notices.

Terminology

The Glossary contains terms and abbreviations that are unique to ABB or have a usage or definition that is different from standard industry usage. Please make yourself familiar to that.

You will find the glossary at the end of the ***Engineering Manual System Configuration.***

Document conventions

The following conventions are used for the presentation of material:

- The words in names of screen elements (for example, the title in the title bar of a window, the label for a field of a dialog box) are initially capitalized.
- Capital letters are used for the name of a keyboard key if it is labeled on the keyboard. For example, press the ENTER key.
- Lowercase letters are used for the name of a keyboard key that is not labeled on the keyboard. For example, the **space bar**, **comma key**, and so on.
- Press CTRL+C indicates that you must hold down the CTRL key while pressing the C key (to copy a selected object in this case).
- Press **ESC, E, C** indicates that you press and release each key in sequence (to copy a selected object in this case).
- The names of push and toggle buttons are boldfaced. For example, click **OK**.
- The names of menus and menu items are boldfaced. For example, the **File** menu.
  - The following convention is used for menu operations: MenuName > MenuItem > CascadedMenuItem. For example: select **File** > **New** > **Type**.
  - The **Start** menu name always refers to the **Start** menu on the Windows Task Bar.

- System prompts/messages are shown in the Courier font, and user responses/input are in the boldfaced Courier font. For example, if you enter a value out of range, the following message is displayed:

  ```
  Entered value is not valid. The value must be 0 to 30.
  ```

  You may be told to enter the string TIC132 in a field. The string is shown as follows in the procedure:

  TIC132

  Variables are shown using lowercase letters.

  sequence name

1 Variables

1.1 General Description - Variables

Variables are used for storing and processing information. Various different data types are available in the system, for example Byte, Word, Integer, Real, Date&Time. To enable several variables to be processed jointly even if they have different data types, it is possible to define **structured data types**. For more information, refer to Structured data types on page 54.

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 or gateway is added. Status details for the resource are stored in these variables.

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

Variables from Freelance 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.

The variable names can consist of letters, digits and the special character "_". A variable name must contain at least one letter or underscore to be able to distinguish variables from constants.

All the variables of the current project are stored and displayed in the **variable list**.

1.2 Data types

1.2.1 Overview of simple 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...4294 967 295	Double integer value without sign	123456787, 4566
UINT	16	0...65 535	Integer value without sign	4000, 66
DWORD	32	0...4294 967 295 (0...2^{32}-1)	Double word	0, 655, 16#0000 0FFF, 8#000 000 000 074, 2#0...0...0...0...0...0...0...0001
WORD	16	0...65 535 (0...2^{16}-1)	Word	2, 554, 16#0FFF, 8#000 004, 2#0000 0000 0000 0001
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
DT	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	+24d20h31m23s647ms ... -24d20h31m23s648ms	Time value	T#22s T#3m30s T#14m7s

(1) IEEE Institute of Electrical and Electronic Engineers; American Association of Experts

For the representation of REAL numbers the following applies: due to the internal mapping only 7 significant digits can be determined during conversion to characters. Very high and very low values are represented in exponential form.

Variables of data type STRING are used to display texts. The variables can be edited e.g. in an FBD program with the **STRING function blocks**. These texts can be used e.g. in the operation log, the SFC criteria 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

1.3 Variable list

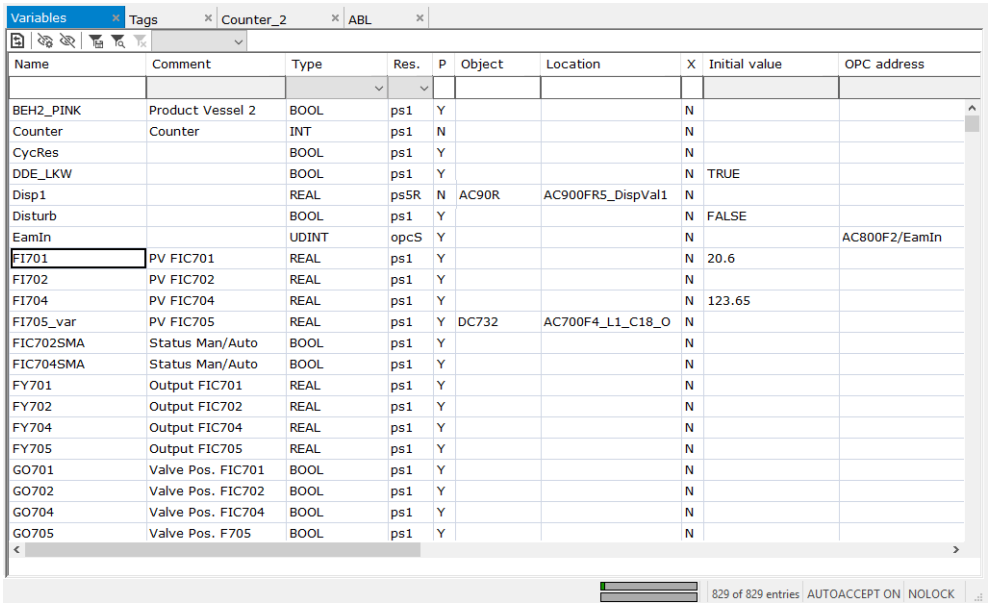
1.3.1 Call the variable list

When opening the project, the variable list is automatically displayed as a separate tab in the right pane. This can be closed and reopened later from the main menu.

> **System > Variable list**

1.3.2 Structure of the variable list

All variables of the current project are displayed in a list.



Name	Comment	Type	Res.	P	Object	Location	X	Initial value	OPC address
BEH2_PINK	Product Vessel 2	BOOL	ps1	Y			N		
Counter	Counter	INT	ps1	N			N		
CycRes		BOOL	ps1	Y			N		
DDE_LKW		BOOL	ps1	Y			N	TRUE	
Disp1		REAL	ps5R	N	AC90R	AC900FR5_DispVal1	N		
Disturb		BOOL	ps1	Y			N	FALSE	
EamIn		UDINT	opcS	Y			N		AC800F2/EamIn
FI701	PV FIC701	REAL	ps1	Y			N	20.6	
FI702	PV FIC702	REAL	ps1	Y			N		
FI704	PV FIC704	REAL	ps1	Y			N	123.65	
FI705_var	PV FIC705	REAL	ps1	Y	DC732	AC700F4_L1_C18_O	N		
FIC702SMA	Status Man/Auto	BOOL	ps1	Y			N		
FIC704SMA	Status Man/Auto	BOOL	ps1	Y			N		
FY701	Output FIC701	REAL	ps1	Y			N		
FY702	Output FIC702	REAL	ps1	Y			N		
FY704	Output FIC704	REAL	ps1	Y			N		
FY705	Output FIC705	REAL	ps1	Y			N		
GO701	Valve Pos. FIC701	BOOL	ps1	Y			N		
GO702	Valve Pos. FIC702	BOOL	ps1	Y			N		
GO704	Valve Pos. FIC704	BOOL	ps1	Y			N		
GO705	Valve Pos. F705	BOOL	ps1	Y			N		

di0347us.png

The status bar shows the number of entries displayed currently. The format is *<entries> of <total entries>*. With activated search filters you can see how many variables meet the search criteria.

The variable list is structured as follows:

Name	Variable name, max. 16 characters
Comment	Comment on variable, max. 33 characters
Type	Data type, see Overview of simple data types on page 22
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 assigned.
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, not directly. This means, the I/O component cannot be read in other resources by using the component name. Note that variables which are to be allocated to an I/O component do not feature gateway write access rights. See also Station access on page 43.

Object, Position

For variables assigned to a hardware component, the component type and slot or the variable is entered here, for example AI723 and AC7_L2_I8 for a channel allocation of an AI 723F module or AC900 and AC9_d_ERR for an AC 900F error signal.

Example AC7_L2_I8:
AC7 Station name in hardware structure
L2 Position of the module in the controller
I8 Component name (channel)

If you double-click one of these two fields, a dialog appears in which you can select a hardware component or variable 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* ☐).

When changing the P attribute, only newly referenced variables will 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 Initial values on page 28.

OPC address

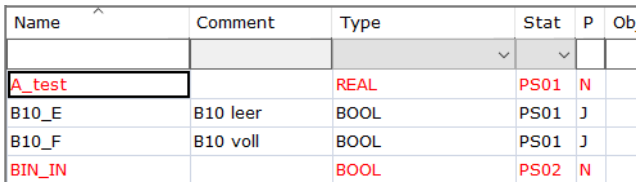
Address or name of a variable on the OPC server. For a Freelance OPC gateway this is identical to the variable name in the process station.

Variables displayed in red either have no references within the project or they may be system variables. See System variables on page 46.

1.3.3 Edit the variable list

Column headers

The arrow ^ in the head of a column indicates that the data is sorted by this column.



header appearance_us.png

Sort list entries

In the variable list, sorting is available for the following columns:

- Name
- Type
- Resource
- Object
- Location

Filter list entries

The user can filter the variables data and save the current filter logic for later use. Based on the type of data, the following are the three kinds of filters available:

Free text filter

The user can type the search criteria in the edit box present below the column header. For example “Name” column filter in variable list.

Drop-down list

The filter criteria are already defined in a drop-down list and the user can select one of them to filter the list. For example “Type” column in variable list.

Save current filter

Once filtering is applied to the list, user has an option to save the current filtering criteria. This preserves the current configuration and can be used in future. This will be project specific. An icon in the toolbar pops up as a dialog, where the user can type a name. This list of saved filters can also be accessed from another toolbar icon. Automatic names are given for filters imported from old projects. The following dialog box appears when you click the **Save filter** from the toolbar.

Call saved filters

User can view, delete and use the list of saved filtering criteria. This list can be opened by clicking the toolbar icon.

Toolbar Icons

Varlist_icons.png

Description of the icons from left to right:

Cross references

The cross references show the places (programs, displays, etc.) in which the selected variable is used.
This icon is enabled only when a variable is selected.

Hide system variables

All variables that have been automatically pre-defined by the system can be shown or hidden.
Click **Hide system variables** to hide the system variables in the variable list.
Click **Hide system variables** again, to show the system variables in the variable list again.

Hide unused variables

All variables that are defined but not used in a program can be shown or hidden.
Click **Hide unused variables**, to hide or unhide the unused variables in the list.

Save filter Saves the current filter settings under a given name. A maximum of ten filter settings can be stored.

Show saved filters

A dialog box opens from which you can select, activate or delete a previously saved filter.

Clear filter Clears all active filter criteria of the variable list. This includes the “Hide system variables” and “Hide unused variables” functions.

Access by gateway station

This drop-down list is used to select one gateway or all gateways. An empty item selection means “all gateways”.The gateway access filter shows only variables that have access to the selected gateway station.

Search in the variable list

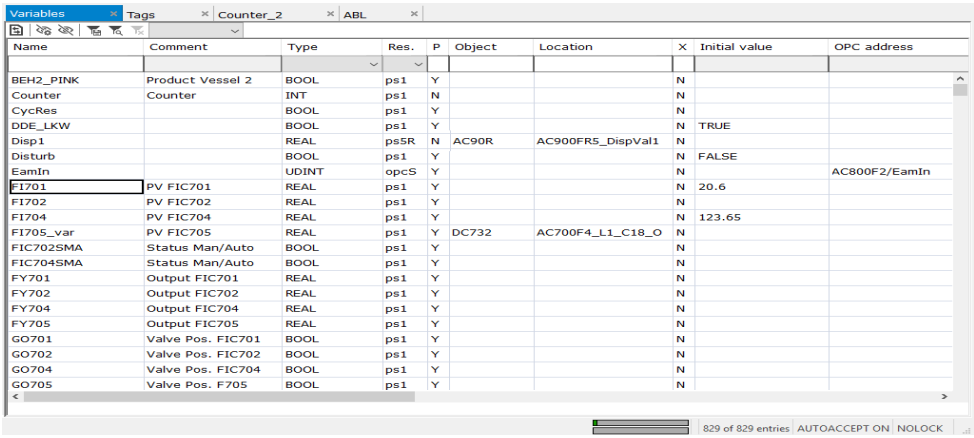
> Edit > Find

The **Find** function allows you to search variables by name. When this function is chosen from the menu or shortcut menu, a dialog containing an input field appears. When a name or the beginning of a name is entered, the list is scrolled automatically until the first matching entry is found.

1.3.4 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.

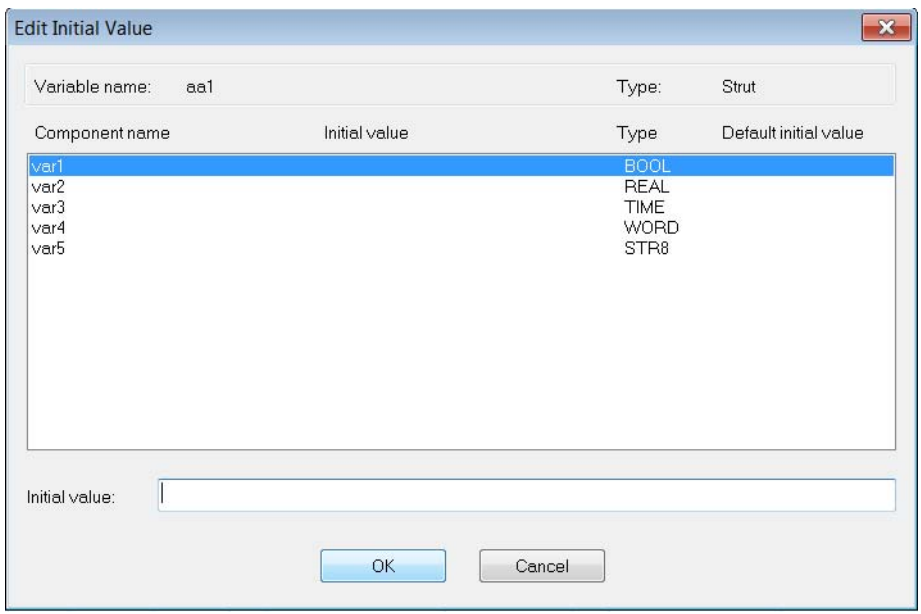
A double-click in the **Initial value** field for a particular variable allows the initial value for that variable to be modified.



di0347us.png

No initial values can be assigned here to variables that are assigned to hardware components.

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 dialog is displayed which shows all the elements of the structured variable's basic data type.



di0346us.png

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 dialog, this is indicated by -**...**- in the variable list.

1.3.5 Normal view and station view

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

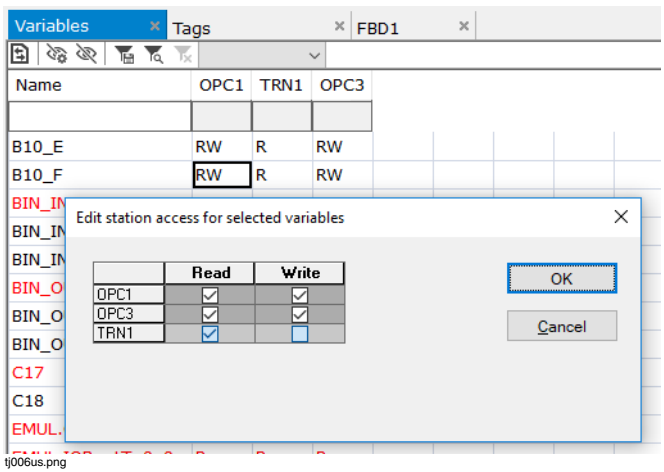
R = Read access - the variable can be read via the gateway.

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

> **Editor > Normal view**

or

> **Editor > Station view**



tj006us.png

> Double-click a resource column

or

> Select a block > **Edit > Station access**

A dialog opens to modify the access rights.

See also Station access on page 43.

1.3.6 Close

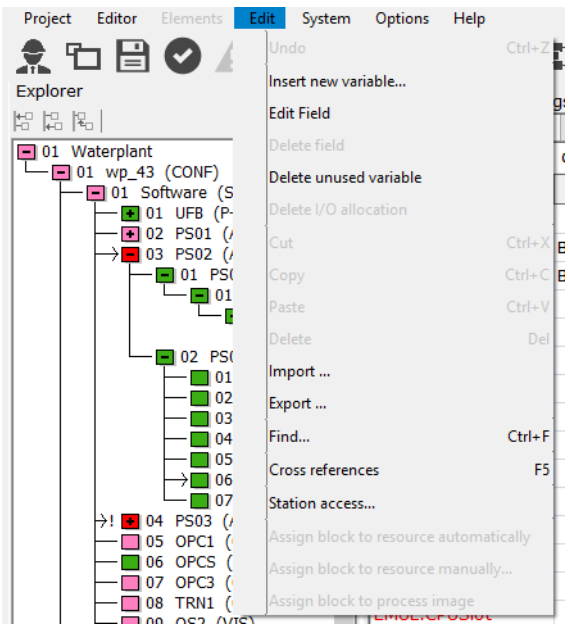
> **Editor > Close**

Closes the **Variables** tab.

1.4 Edit variable list entries

> **Edit**

Various menu options are available for editing the individual list entries. For example, the last action can be undone, new entries can be inserted, entries can be deleted, cut or copied. Blocks or variables can be imported and exported.



di0338us.png

1.4.1 Undo

> **Edit > Undo**

The last change is undone and the old status restored. If it is not possible to undo the last action, the menu item is disabled.

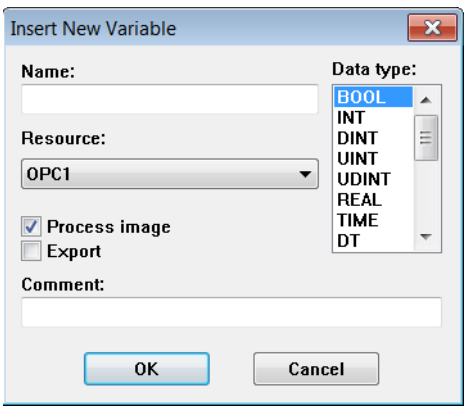
1.4.2 Create a new variable in the list

> **Edit > Insert new variable**

When any filter is activated, i.e. the list is not fully displayed, it is not possible to insert a new variable.

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

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.



di0335us.png

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 cannot be written via a gateway.

Comment Comment in the form of free text.

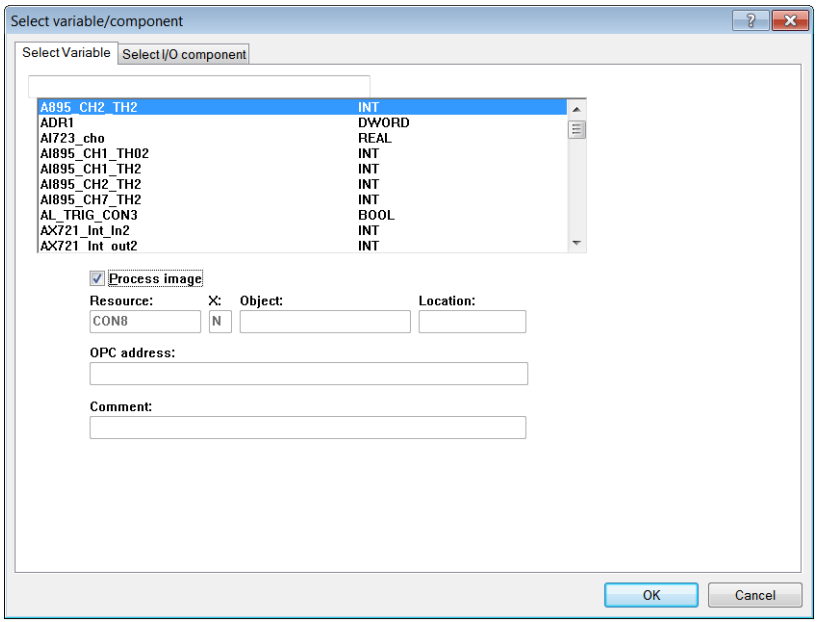
When any filter is activated, that is the list is not fully displayed, it is not possible to insert a new variables.

1.4.3 Create a new variable in a 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.

1.4.4 Insert an 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. In the following dialog, a variable already defined in the project can be selected for use.



di0345us.png

Variable via Process image

A choice can be made as to whether the variable can be read from the process image. Refer to ***Engineering Manual, System Configuration, Project tree***.

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

1.4.5 Edit a field in the list

> Select field by double-click. The cursor is positioned at the last entry position

or

> **Edit > Field**

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

Changing existing variables may affect other programs. In order to avoid errors a list of the affected programs is displayed when changes are made. A decision can be made as to whether or not the changes are to be carried out.

1.4.6 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.

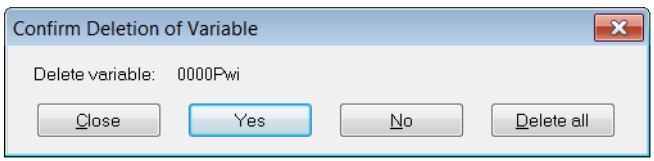
> Click the desired field > **Edit** > **Delete**.

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 mouse button, and lastly removing the text thus selected by pressing the **Delete** button.

1.4.7 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**



di0340us.png

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	Aborts the delete function.

Variables for which access rights have been assigned via a gateway, but which are not used in any program, are considered as unused variables.

1.4.8 Delete I/O allocation

The hardware allocation, **object** and **position** entries, of the selected variables are deleted.

1.4.9 Edit block

Only one block can be defined in each case respectively. A block consists of a set of consecutive rows in the list and can be selected as follows:

> Click cursor where the block is to start

> Press left mouse button and drag the mouse to the end of the block to mark it

or

> Press **SHIFT** key and move cursor using arrow keys

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

Cut

> Select block > **Edit** > **Cut**

The 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.

Copy

> Select block > **Edit** > **Copy**

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

Paste

> Select block > **Edit** > **Paste**

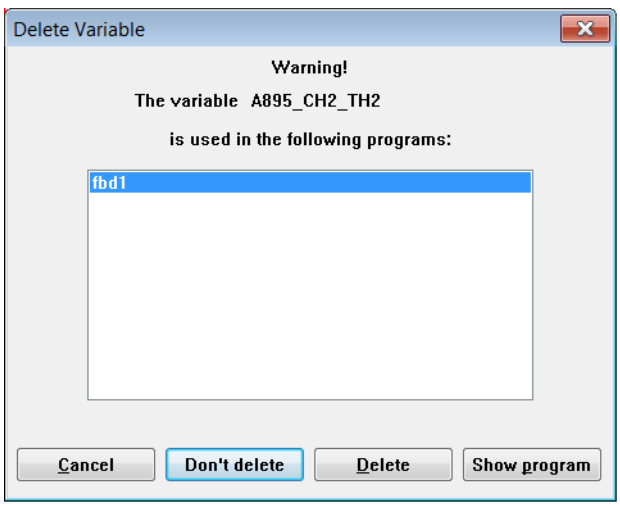
A copied or cut block in the clipboard is inserted at the position defined by the cursor.

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

Delete

> Select block > **Edit** > **Delete**

A warning message with a query for confirmation will appear for each variable which is still used in other programs.



di0348us.png

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

1.4.10 Export

> Select one or more variables in the list > **Edit** > **Export...**

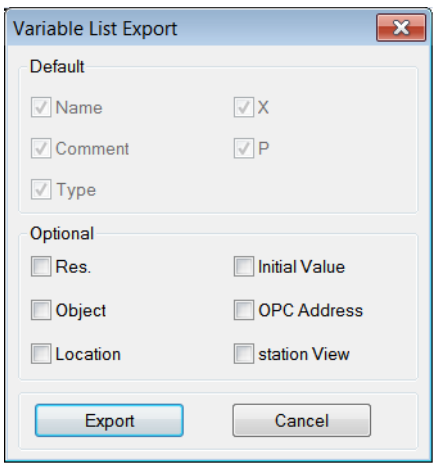
> Select the desired file type ***.eam or *.csv** > specify file name

The selected entries are saved as a file on a data medium (hard disk). An additional window appears in which the file path and file name must be entered. This file may be imported into other projects via the menu item **Import...**.

Data for resource (Res.), object and location can not be imported.

Two file types are available for the export; the Freelance file format with the extension **EAM** and the external file format **CSV** (comma separated values) which can be read by external applications like Microsoft Excel.

For an export to a CSV file the user has to specify which information of the selected tags should be exported.



Export_var_us.png

The standard information items **Name**, **Comment**, **Type**, **Export flag** and **Process image flag** are mandatory an cannot be de-selected.

Optionally, the parameters **Resource**, **Object**, **Location**, **Initial value**, **OPC address** and **Station view** can be selected.

The first row of the CSV file contains the column headers, from the second row onwards the variable list information is stored.

Example for a CSV file if all options are selected:

```
Name, Comment,Type, Res. X;Object;Location;P;Initial value;OPC address;OPC;trn
LI700SL1;Status Limit1 LI700;BOOL;ps1;N;AC900F;AC900F2_ERR;;Y;;;R;R
LI700_var;Level T-100;REAL;ps1;N;;;Y;;;RW;R
LI704;Process Value LIC704;REAL;ps1;N;;;Y;;;;R
LI750_3_var;Actual Level REA1;REAL;ps1;N;;;Y;;;R;R
```

CSV_file_vars_us.png

Name	Name of the variable
Comment	Comment of the variable
Type	Data type
Res.	Name of the associated resource
X	Export: "Y" or "N" - Variable is exported to other resources via lateral communication or not.
Object	Name of the assigned hardware component
Location	Component name of the assigned hardware component
P	“Y" or "N" - Variable is read via process image or not
Initial value	Configured initial value
OPC address	Name of the OPC item, if this variable is read from an OPC server
OPC;trn	Resource names of the gateway stations in the project R = Variable can be read via this gateway station. RW = Variable can be read and written via this gateway station. (empty) = Variable cannot be accessed via this gateway station.

1.4.11 Import

If variables are defined outside of Freelance Engineering, either in another Freelance project or with an external application, these variables can be imported from a file into the project.

Two file formats are supported: the Freelance file format with the extension **EAM** and the external file format **CSV** (comma separated values).

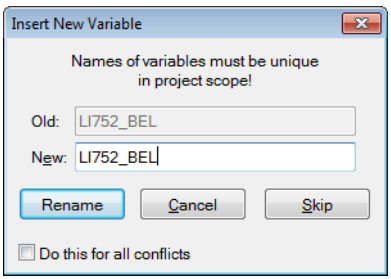
An EAM file was created via Export from a variable list of Freelance Engineering.

A CSV file was created by an external application or with a text editor. The entries are separated with a semicolon ';'. If a text item itself includes the list separator, the text entry should be enclosed in quotation marks (" "); for example "xxx;xxx". The end of the file is marked by a line break.

The first row of the CSV file contains the column headers, the second row onwards contains the variable list information. See also description of **Export** above. At least the column "Name" must exist to import the CSV file, all other columns are not mandatory.

> **Edit** > **Import** > select file type ***.eam or *.csv** > select file

If a variable should be imported with a name that already exists in the project, a dialog is shown:



nsert_new_variable_US.png

If the user should not enter a new name and press **Rename**, the record will be overwritten with the information of the CSV file. Fields that are not specified in the CSV file will not be modified via the import. The overwrite operation must be confirmed by the user.

Overwriting columns of an existing variable record is possible only when importing a CSV file. Existing data for object and location will be deleted. With the import from an EAM file all new variables must get unique names. Data for resource (Res.), object and location can not be imported.

If the user enters a new name in the renaming dialog, the data record of the existing variable is copied and modified with the information from the file. Thus, omitted fields in the CSV file will assume the value of the original variable.

If no naming collision is found, a new variable is created with the information of the file. For the omitted fields default values are used:

Column name	Default value
Comment	“
Data Type	Others
Resource	(------)
Export	N
Process	Y
Initial Value	0/""
Station View	R

If a value which is tightly coupled with system values, for example data type, has an invalid entry, the value will be replaced with default value. Any columns other than the described fields above will be ignored.

Press the **Skip** button to ignore the current variable and continue with the next entry in the file.

During the import, a ".log" extension files is created with the same path and name as the imported CSV file. In this LOG file the import errors are listed and also those variables that could not be imported automatically, together with the information **Invalid**, **Skip** and **Rename**.

1.4.12 Cross references

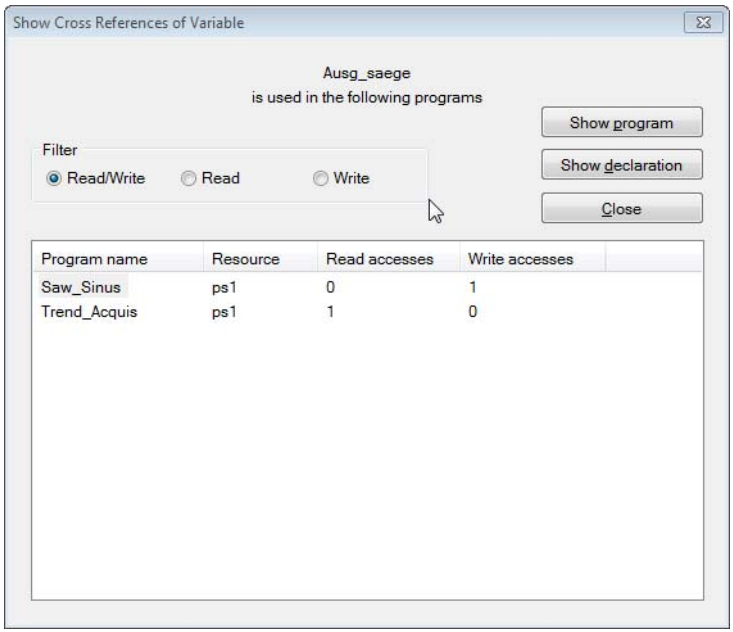
All cross references for a variable can be shown in a list. Cross references are references to this variable in programs, displays, logs and so on, in other words to places where this variable is used.

> Select field > **Cross references** or **F5** key

or

> **Edit** > **Cross references**

A window displays the names of affected programs and the information whether the variable is read or written by these programs.



tj007us.png

Show program

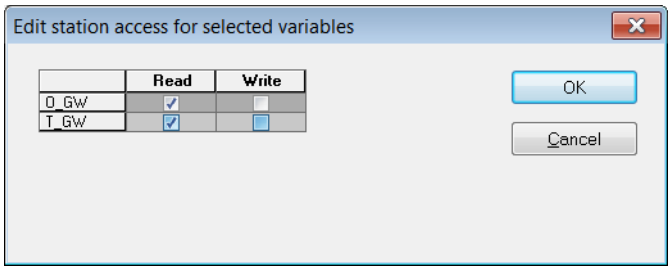
Calling a program with pre-selection of this variable or calling the module to which the variable is allocated.

Show declaration

Variable list remains selected, the selected variable is marked.

1.4.13 Station access

> Select block > **Edit** > **Station access**



di0344us.png

If the variable is to be read or written through a gateway station this access must be enabled in the following items:

- in the project tree on the gateway station
- in the variable list

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

Call up the **Station view** of the variable list to get an overview of the access rights for all variables.

For more information, see also Normal view and station view on page 30 and ***Engineering Manual Freelance OPC Server***.

1.4.14 Assign block to resources 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 block to resources manually** menu can be used to subsequently manually assign those variables which proved impossible to assign automatically. For more information, refer to Assign block to resources manually on page 44.

> Select variable or block

> **Edit > Assign block to resources automatically**

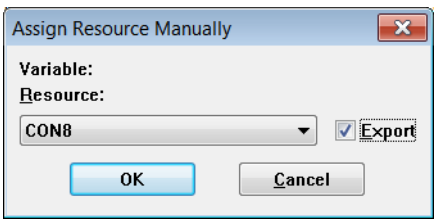
The resource is assigned and entered in the Res column.

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

1.4.15 Assign block to resources manually

> Select variable or block

> **Edit > Assign block to resources manually**



di0352us.png

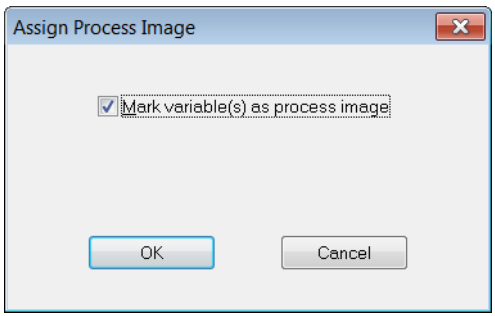
Each variable should be assigned to precisely one resource (process station). Following a block import none of the variables that have been newly added to the project database during the import process have yet been allocated to a resource. Variables which already existed in the project retain their 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 ticked, variables from other resources can be read in.

1.4.16 Assign block to process image

> Select variable or block

> **Edit** > **Assign block to process image**



tj004us.png

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

1.5 Options

1.5.1Print

> **Options** > **Print**

The contents of the screen are output to the printer.

1.5.2Adjust colors

> **Options** > **Colors**...

The color of unused variables can be specified.

1.5.3Save column settings

> Options > Save column settings

The column width setting is saved.

1.5.4Auto accept

Turning on/off auto save

> Options > Auto accept

Select **Auto accept** to automatically save any changes done in the current editor before switching to another editor.

1.5.5Save filter

Saves the current filtering under a given name. A maximum of ten filter settings can be stored.

1.5.6Clear filter

Clears all active filter criteria of the variable list. This includes the “Hide system variables” and “Hide unused variables” functions.

1.5.7Show saved filters

A dialog opens from which you can select, activate or delete previously saved filter settings.

1.6 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, that is 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 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 process station resource
column **G**: X = system variable of a gateway station resource

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

1.6.1 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.CPatchVerNov	UINT	X	X	Current version number for project “amendments”. It increases every time the function block is changed

1.6.2 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 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 in Kilo Bytes.
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 Freelance Engineering
xxxx.MsrStopped	BOOL	X		Boolean variable, logic = 1 when the station is shutdown via a RUN/STOP switch on the CPU module

Variable name	Data type	P	G	Designation
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
xxxx.Configuring	BOOL	X		Boolean variable, logic = 1 when the station is being configured by Freelance Engineering
xxxx.EMajorVerNo	UINT	X	X	Current major EPROM version number
xxxx.EMinorVerNo	UINT	X	X	Current minor EPROM version number
xxxx.CPURack	UINT	X		ID of the rack which the currently active CPU module (Primary CPU) is plugged into.
xxxx.CPUSlot	UINT	X		Slot holding the currently active CPU module (Primary CPU).

Variable name	Data type	P	G	Designation
xxxx.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 by configuration in Freelance Engineering (enable external time synch.)

1.6.3 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).

Variable name	Data type	P	G	Designation
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.

1.6.4 System variables for power fail on voltage failure

Variable name	Data type	P	G	Designation
xxxx.NoPowerFail	UINT	X		Present number of Power Fails which did not lead to a warm start. The variable is initialized at zero after a cold start
xxxx.PowerOffTim	TIME	X		Only for AC 800F and DCP Controller. 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.

1.6.5 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 "not executable". Refer to ***Engineering Manual Process Station, Task error messages***.
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.

1.6.6 System variables for I/O communication

These variables are used only with rack-based I/O modules.

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	53	DCP 02a, new hardware revision of DCP 02
11	DDI 04, 28 x Namur initiators or 12 x 3/4-wire initiators	56	DCP 10, gateway
12	DDI05, 32 x 120/230 V AC	60	DDO 02, 16 x 230 V AC/DC

20	DDO 01, 32 x 24 V DC, 0.5 mA	61	DDI 02, 16 x 24..60 V AC/DC
30	DAI 01, 16 x 0/4..20 mA, 50 Ohm	62	DDI 03, 16 x 90..230 V AC
31	DAI 02, 16 x 0..10 V DC	63	DDO 03, 16 x 24..60 V AC/DC, read back
32	DAI 03, 16 x 0/4..20 mA, 250 Ohm	64	DDO 04, 16 x 115..230 V AC, read back
35	DAI 05, 16 x 0/4..20 mA, MU powering	70	DAI 04, 8 x PT100/mV
40	DAO 01, 16 x 0/4..20 mA, R_I=400 Ohm	80	DFI 01, 4 x f <= 45 kHz
50	DCP 02, CPU	89	DLM 01 - Link module
51	DCP 10, CPU	90	DLM 02 - Link module
52	DCP 02, gateway	100	DCO 01, 4 x RS 485/422/232 C

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.

1.6.7 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.

1.7 Structured data types

Application-specific data types can be created, that is 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 through 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.

1.7.1 Call definition of structured data types

> **System** > **Structured data types**

1.7.2Define 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**

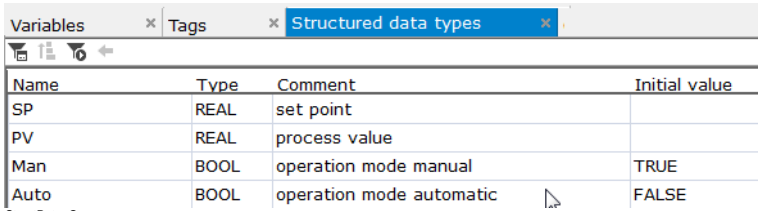
or

> Double-click the name field of the first free line.

1.7.3Create data type components

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

> **Define!**



StructData_Comp_us.png

A name and an elementary data type are entered for each component which is to be available under the new data type.

Name Name of the component max. 16 characters

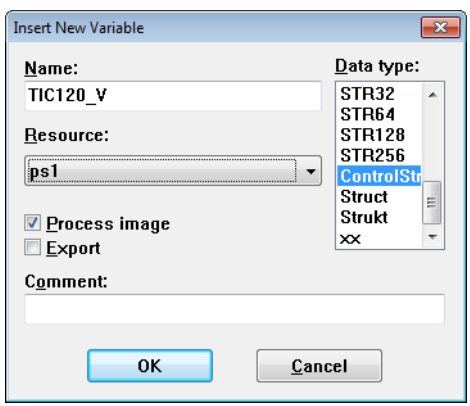
Type Data type, for example BOOL, INT or REAL.
See also Overview of simple data types on page 22.

Comment Any text to describe the entry.

Initial value The default initial value for a single element is specified here for all instances of this structured variable, but may be reassigned in the variable list for each usage of this data type. See also Initial values on page 28.

1.7.4 Insert a new variable with structured data type

> System > Variable list > Edit > Insert new variable



StructData_Var_us.png

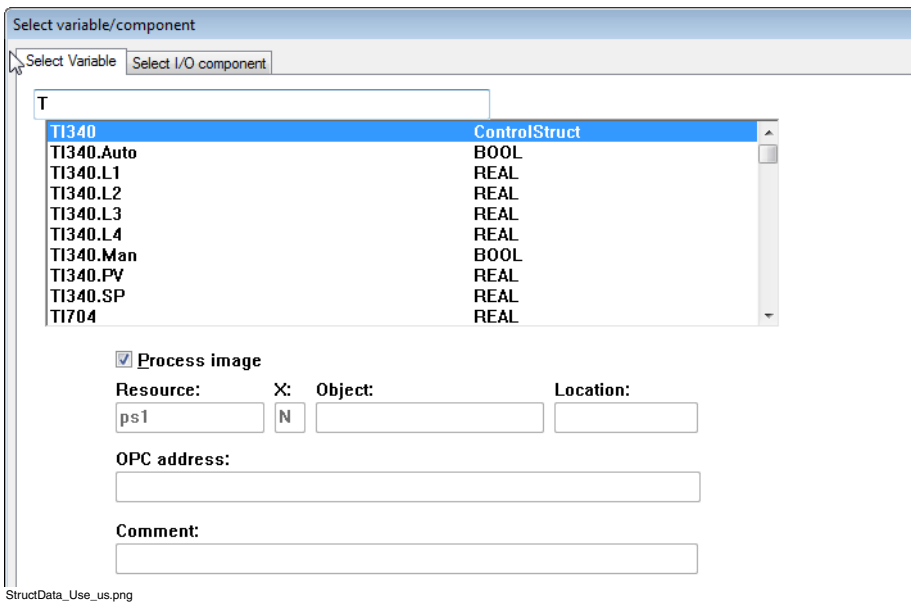
Using the new data type **ControlStruct** the corresponding variables can be declared. For example, to supply multiple controllers of the same type, only one variable of the new type **ControlStruct** must be created for each controller. Now, all components are available with their elementary data types for this structured variable.

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

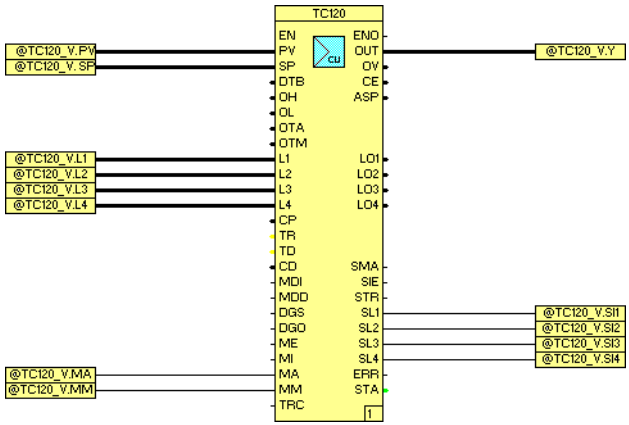
TC120_V.SP	REAL	Set point
TC120_V.PV	REAL	Process value
TC120_V.Man	BOOL	Operation mode **Manual**
TC120_V.Auto	BOOL	Operation mode **Automatic**

1.7.5 Use a structured variable in a program

> For example, select a read or write variable in a function block diagram.



StructData_Use_us.png



di0349uk.png

In the window shown above various components of the structured variable TC120_V (data type **ControlStruct**) have been used.

2 Tags

2.1 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.

This list is 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).

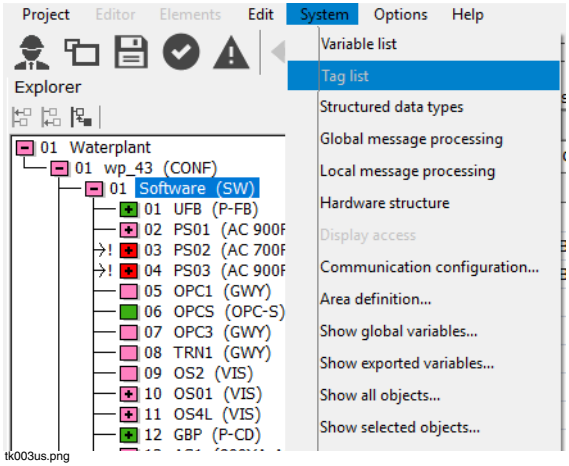
The maximum length of tag names can be configured for a project with 12 or 16 characters. See also ***Engineering Manual System Configuration, Project manager***.

Search criteria can be defined and activated. The status bar shows the currently displayed number of entries. The format is <entries> of <total entries>. When search filters are active, this enables you to see how many tags meet the search criteria.

2.1.1 Call the tag list

> System > Tag list

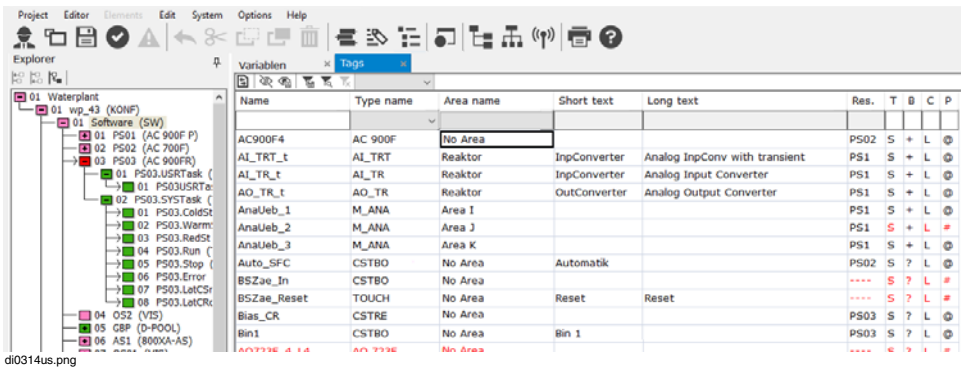
When opening the project, the tag list is automatically displayed as a separate tab in the right pane. This can be closed and reopened later from the main menu.



tk003us.png

2.1.2 Structure of the tag list

All tags of the current project are displayed in a table.



di0314us.png

The status bar shows the number of currently displayed entries. The format is <*entries*> *of* <*total entries*>. When search filters are active, this enables you to see how many tags meet the search criteria.

Name Tag name , max. 12 or 16 characters. For details see ***Engineering Manual System Configuration, Project manager***

T Object type of entry:
S Standard name.
Name of a function block, name of an SFC program, name of a module or an object in the hardware structure; also 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'.

Res Resource name

Area name Plant area which is assigned to the tag.

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

R State of processing, cannot be modified in this list
+ Processing of the block is enabled (Processing ☑),
- Processing of the block is not enabled (Processing ☐),
? Processing of function not defined (Processing ▢),

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

For templates from the hardware structure 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).

T OPC function block class
X FF function blocks

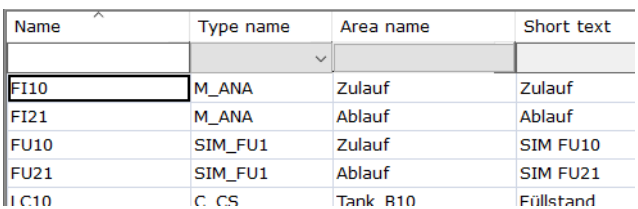
P Status of the plausibility check:
Configuration for the block is not correct.
Configuration errors were reported from last check for this block.
@ Plausibility check for the block without error.
No configuration errors were reported from the last check for this block.

See also ***Engineering Manual System Configuration, Project tree***.

2.1.3 Edit the tag list

Column headers

The arrow ^ in the head of a column indicates that the data is sorted by this column.



Name	Type name	Area name	Short text
FI10	M_ANA	Zulauf	Zulauf
FI21	M_ANA	Ablauf	Ablauf
FU10	SIM_FU1	Zulauf	SIM FU10
FU21	SIM_FU1	Ablauf	SIM FU21
LC10	C_CS	Tank_B10	Füllstand

Tags_Appearance_us.png

Sort list entries

In the tag list, sorting feature is available for the following columns:

- Name
- Area name: Sorting is based on indexes starting from A, B, C, ...N, O. Sorting is not based on the Area names given by the user.
- Type name

Filter list entries

The user can filter the data, and save the current filter logic. Based on the type of data, the following are the three kinds of filters available:

Free text filter

The user can type the search criteria in the edit box present below the Column header. for example, “Name” column filter in the tag list

Drop-down list

The Filter criteria are already defined in a drop down list and user can select one of them to filter the list. for example, “Type name” column in tag list.

List with multiple selection

Allows the user to select multiple items from a given list, for example “Area” column in tag list.

Save current filter

Once filtering is applied to the list, user has an option to save the current filtering This preserves the current configuration and can be used in future. This will be project specific. An icon in toolbar pops up as a dialog, where user can type a name. This list of saved filters can also be accessed from another toolbar icon. Automatic names are given for filters imported from old projects.

Call the saved filters

User can view, delete and use the list of saved filtering criteria. This list can be opened by clicking the toolbar icon.

Toolbar Icons

Below the **Tag** tab, toolbar will have the following icons.

Taglist_icons.png

Description of the icons from left to right:

Cross references

The cross references show the places (programs, displays, etc.) in which the selected tag is used.
This icon is enabled only when a tag is selected.

Hide unused tags

All tags that are defined but not used in the project can be shown or hidden.
Click **Hide unused tags** to hide or show the unused tags in the list.
Show only tags with faceplates
All tags which are without faceplates can be shown or hidden. Click this icon to hide or show all tags without faceplates.

Save filter Saves the current filter settings under a given name. A maximum of ten filter settings can be stored.

Show saved filters

A dialog box opens from which you can select, activate or delete a previously saved filter.

Clear filter Clears all active filter criteria of the tag list. This includes the function “Hide unused tags” and “Show only tags with faceplates”.

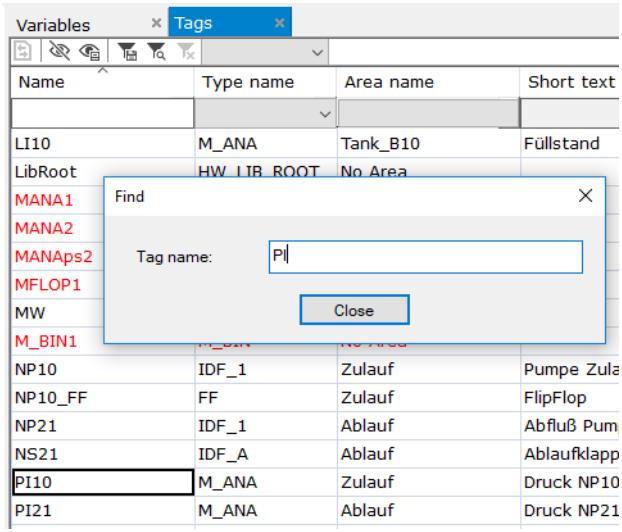
Access by gateway station

This drop-down list is used to select one gateway or all gateways. The empty item selection means “all gateways”.The gateway access filter shows only tags that have access to the selected gateway station.

Search in the tag list

> Edit > Find

The **Find** function allows you to search tags by name. When this function is chosen from the menu or the shortcut menu, a dialog with an input field appears. When a name or the beginning of a name is entered, the list scrolls automatically and the first matching entry is shown.



tk0440us.png

2.1.4 Normal view and station view

In addition to the normal view, a station view can also be selected. In the station view, parameters are set for each tag to define whether they can be read and/or written via a gateway and whether they can be accessed from the operator stations.

R = Read access - the tag can be read via the gateway.

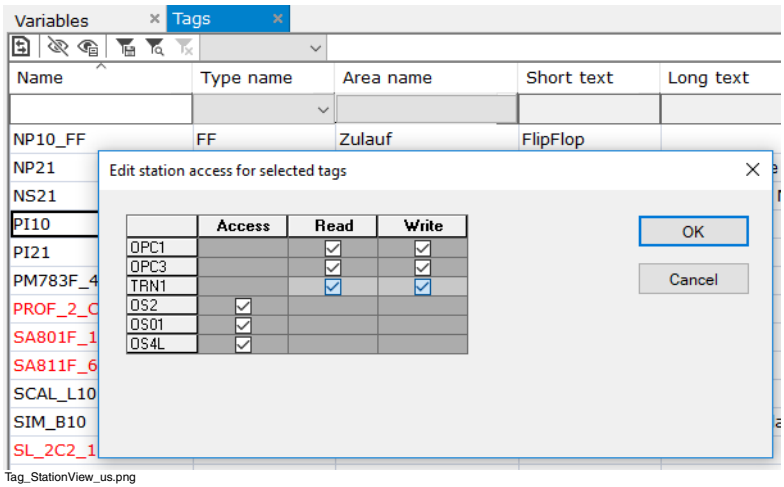
W = Write access - the tag can be written via the gateway.

X = Operation access - tag can be accessed from the operator station..

> **Editor> Station view**

or

> **Editor > Normal view**



Tag_StationView_us.png

> Double-click a resource column

or

> Select a block > **Edit > Station access**

A dialog opens to modify the access rights.

See also Station access on page 76 and ***Engineering Manual Freelance OPC Server***.

2.1.5 Close

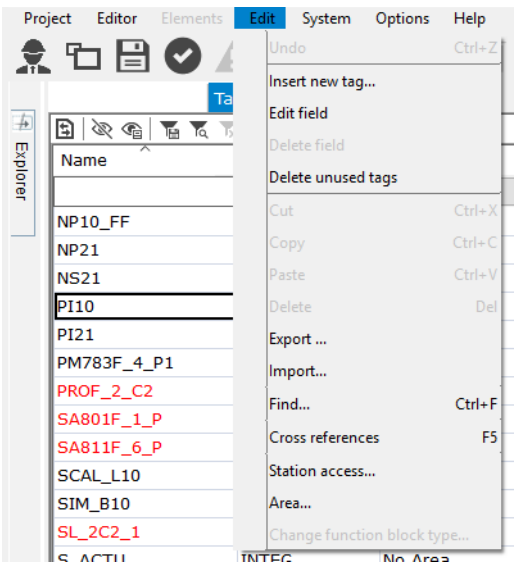
> Editor > Close

Closes the tag tab.

2.2 Edit tag list entries

> **Edit**

Various menu options are available for editing the individual list entries. For example, the last action can be undone, new entries can be inserted, entries can be deleted, cut or copied. Blocks or tags can be imported and exported.



di0322us.png

2.2.1 Undo

> **Edit > Undo**

The last change is undone and the old status restored. If it is not possible to undo the last action, the menu item is disabled.

2.2.2 Insert new tag in the list

> Edit > Insert new tag

When any filter is activated, that is the list is not fully displayed, it is not possible to insert a 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 be changed then by entering the desired new name. All the other data is taken over from the tag which was selected previously.

Tagname_unique_us.png

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.
Rename	If a new unique name has been defined, a new tag is created-
Cancel	The existing tag is not changed.
Skip	Closes the dialog, no changes are done. This button is mainly used for importing a set of tags from a file; refer also to Import on page 73.

2.2.3 Edit a field in the list

> Double-click to select field and position cursor at the last entry position

or

> **Edit** > **Edit field**

> Enter changes

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

Changing existing tags may affect other programs. In order to avoid errors, a list of the affected programs is displayed when changes are made. A decision can be made as to whether or not the changes are to be carried out. See also Cross references on page 75.

2.2.4 Delete field

Only the fields **Short text** and **Long text** can be deleted with this command.

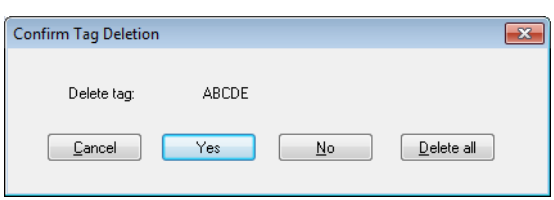
If a whole line in the list is selected, then the tag may be deleted

> Click field **> Edit** > **Delete**

2.2.5 Delete unused tags

All entries with no cross-references (these tags are identified by a red color) are deleted following a query for confirmation.

> **Edit** > **Delete unused tags**



Confirm Tag Deletion_us.png

Yes	The tag that is displayed is deleted.
Delete all	All unused tags (all tags in red) are deleted.
No	The tag that is displayed is not deleted, and the next tag is displayed.
Cancel	Aborts the delete function.

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

2.2.6 Edit block

Only one block can be defined in each case respectively. A block consists of a set of consecutive rows in a list and can be selected as follows:

> Click cursor where the block is to start

> Press left mouse button and drag the mouse to the end of the block to mark it

or

> Press **SHIFT** key and move cursor using arrow keys.

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

Cut

> Select block > **Edit** > **Cut**

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

Copy

> Select block > **Edit** > **Copy**

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

Paste

> Select block > **Edit** > **Paste**

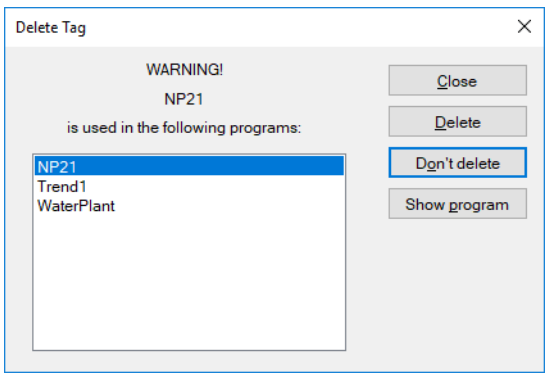
A copied or cut block in the clipboard is inserted at the cursor position.

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

Delete

> Select block > **Edit** > **Delete**

For each tag which is still used in other programs, a warning message will appear with a request for confirmation.



di0355us.png

Close Back to the corresponding list

Don't delete Selected variable/tag is not deleted

Delete Selected variable/tag is deleted

Show program
Go to selected program

2.2.7 Export

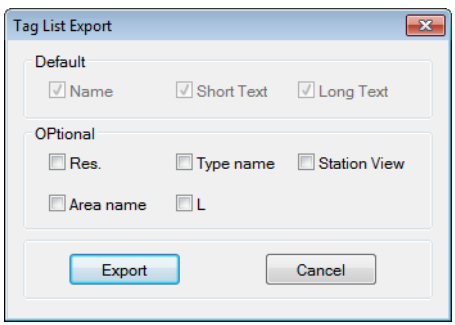
> Select one or more tag entries in the list > **Edit** > **Export**

> Select the desired file format ***.msr** or ***.csv** > specify file name

The selected entries are saved as a file on a data medium (hard disk). An additional window appears into which the file path and file name must be entered. This file may be read into other projects via the menu item **Import...**.

Two file types are available for the export; the Freelance file format with the extension **MSR** and the external file format **CSV** (comma separated values) which can be read by external applications like Microsoft Excel.

For an export to a CSV file the user has to specify which information of the selected tags should be exported.



Tag_list_export_us.png

The standard information **Name**, **Short text** and **Long text** are mandatory an cannot be de-selected.

Optionally the parameters **Resource**, **Area**, **Type name**, **Library type** and **Station view** can be selected.

The first row of CSV file contains the column headers, from the second row onwards the **Tag** list information is stored.

Example for a CSV file if all options are selected:

```
Name, Res; plant area; short text; Long Text, Type Name C; V_GR; V_US; OPC; trn
LI328us; ps1; Vessel 71; Scale dosing;; DOS_S, D, X, X, RW; R
LI700_1; ps1; reactor; reset and reset the alarm level to high 40s; TON, L, X, X, RW; R
```

CSV_file_tags_us.png

Name	Name of the tag entry
Res.	Name of the associated resource
plant area	Name of the plant area assigned to the tag
short text	Configured short text of the tag
Long Text	Configured long text of the tag
Type Name	Short name of the tag type
C	Library type, see Structure of the tag list on page 60
V_GR; V_US	Resource names of the operator stations in the project X = Tag can be accessed from this operator station (empty)= tag cannot be accessed from this operator station
OPC;trn	Resource names of the gateway stations in the project R = Tag can be read via this gateway station RW = Tag can be read and written via this gateway station (empty)= Tag cannot be accessed via this gateway station.

2.2.8 Import

If tags are defined outside of Freelance Engineering, either in another Freelance project or with an external application, these tags can be imported from a file into the project. Three file formats are supported, the Freelance file format with the extension **MSR** and the external file formats **CSV** (comma separated values) and **TXT** (text file); the content of the CSV and the TXT file is identical.

An MSR file was created via Export from a tag list of Freelance Engineering.

A CSV file (or TXT file) was created by an external application or with a text editor. The entries are separated with a semicolon ‘;’. If a text item itself includes the list separator, the text entry should be enclosed in quotation marks (“ ”); for example "xxx;xxx". The end of the file is marked by a line break.

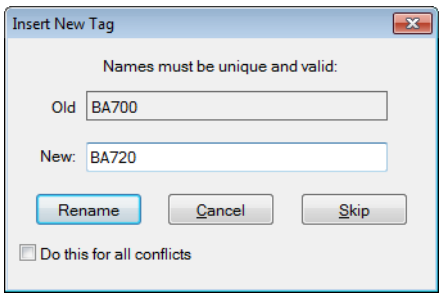
The first row of the CSV file contains the column headers, the second row onwards contains the tag list information. See also description of Export above.

At least the column "Name" must exist to import the CSV file, all other columns are not mandatory.

If the header row does not exist in the CSV file, the first three entries of each line are imported as Name, Short text and Long text. All other entries in the file are ignored.

> **Edit > Import... >** select file type ***.msr**, ***.csv** or ***.txt** > select file

If a tag should be imported with a name that already exists in the project, a dialog is shown:



Insert_new_tag_us.png

If the user does not enter a new name and press **RENAME**, the record will be overwritten with the information of the CSV file. Fields that are not specified in the CSV files will not be modified by the import. The overwrite operation must be confirmed by the user.

Overwriting columns of an existing tag record is possible only with import of a CSV file. With the import from MSR files all new tags must get unique names.

If the user enters a new name in the renaming dialog, the data record of existing tag is copied and modified with the information from the file. Thus, omitted fields in the CSV file will get the value of the original tag.

If no naming collision is found, a new tag is created with the information of the file. For the omitted fields default values are used:

Short Text	“”
Long Text	“”
Resource	(-----)
Area Name	No Area

Short Text	“’
Type Name	“”
Station view	R/””

If a value which is tightly coupled with system values, for example data type, has an invalid entry, the value will be replaced with the default value. Any columns other than the fields described above will be ignored.

Press the **Skip** button to ignore the current tag and continue with the next entry in the file.

During the import, a ".log" extension files is created with the same path and name as the imported CSV file. In this LOG file the import errors are listed and also those tags that could not be imported automatically with the information **Invalid**, **Skip** and **Rename**.

2.2.9 Cross references

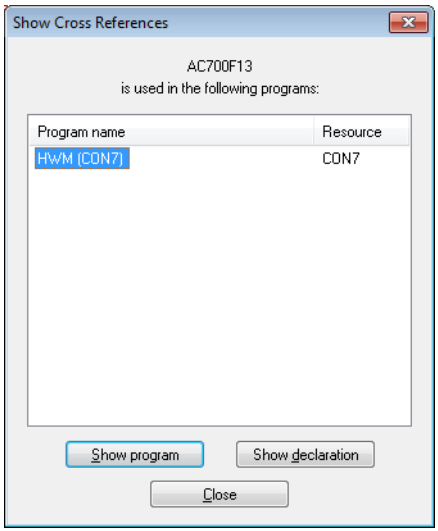
Cross references of a tag can be shown in a list. Cross references are references relating to this tag in programs, displays, listings and so on, in other words to places where this tag is used.

> Select field > **Cross references** or **F5** key

or

> **Edit** > **Cross references**

A window displays the names of the relevant programs:



di0356us.png

Show program

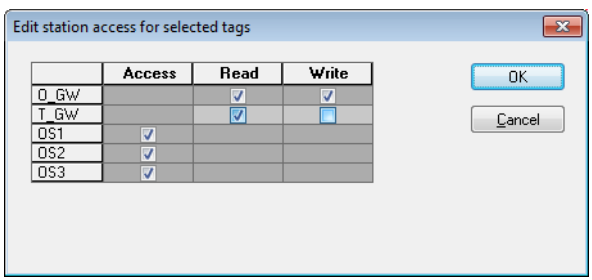
Calling a program with pre-selection of this tag or calling the module to which the tag is allocated

Show declaration

Tag list remains selected, the selected tag is marked.

2.2.10 Station access

> Select block **> Edit > Station access**



di0321us.png

If the inputs, outputs and parameters of a tag are to be read or written via a gateway, this access must be enabled in the following items:

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

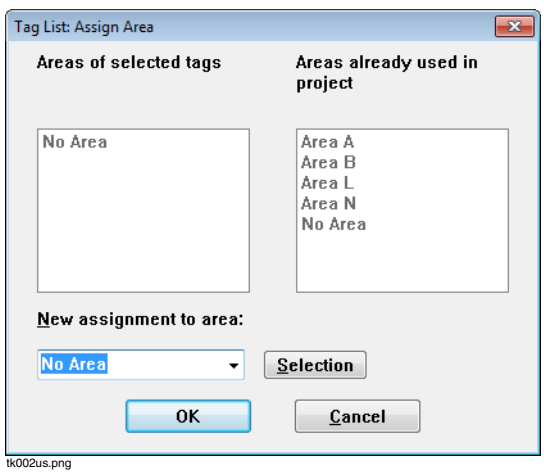
For each operator station, certain tags which should not be operated on this station can also be filtered. If no access is released, this tag cannot be selected from the tag list of this operator station.

Call up the **Station view** of the tag list to get an overview of the access rights for all tags.

For more information, see Normal view and station view on page 65 and ***Engineering Manual - Freelance OPC-Server.***

2.2.11 Area

> Select block > **Edit** > **Area**



tk002us.png

Areas of selected tags

All areas already assigned in the selected block are shown.

Areas already used in project

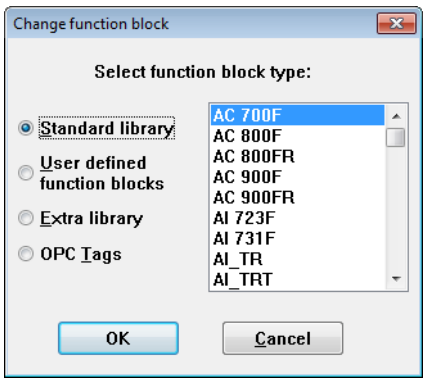
All areas that are used in the whole project are listed.

Selection All the tags in the selected block are assigned to the plant area entered here.

2.2.12 Change function block type

> Select block > **Edit** > **Change function block type**

A new block type can be allocated to the marked tags. All block types of all libraries known by the system are available for selection.



tk002us_1.png

2.2.13 Access rights

> Select block > **Edit** > **Access rights**

If the add-on package Security Lock 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 observed, or also operated, or not called at all.

Refer to ***Engineering Manual User Access***.

2.2.14 User groups

> Select block > **Edit** > **User groups**

If the add-on package Security Lock is installed, individual user groups can be assigned here to certain resources. Refer to ***Engineering Manual User Access***.

2.3 Options

2.3.1Print

> **Options** > **Print**

The screen contents may be output to a printer.

2.3.2Adjust colors

> **Options** > **Colors...**

The color for unused tags can be specified.

2.3.3Save column settings

> **Options** > **Save column settings**

The column width setting is stored.

2.3.4Auto Accept

Turning on/off auto save

> **Options** > **Auto Accept**

Select Auto Accept to automatically save any changes done in the current editor before switching to another editor.

2.3.5Save filter

Saves the current filtering under a given name. A maximum of ten filter settings can be stored.

2.3.6Clear filter

Clears all active filter criteria of the variable list. This includes the “Hide system variables” and “Hide unused tags” functions.

2.3.7Show saved filters

A dialog box opens from which you can select, activate or delete a previously saved filter setting.

3 OPC items

3.1 General Description - OPC items

OPC items represent the connection to process variables and tags which will be provided by an OPC server. The standardized OPC interface gives the possibility to connect different control systems with each other. The OPC items can be of two types:

- Data Access (DA) items
- Alarm & Events (AE) items

The **OPC items** dialog is used for easy integration of OPC items into a Freelance system by Freelance Engineering. If a connection is established between Freelance Engineering. and the OPC server of another system, the configuration of the OPC server can be read through the browser interface. The read OPC items are displayed in a list view.

OPC items (DA and AE) can be used to define a new function block type (OPC_FB-CLASS). A faceplate can be configured for each function block class. In a second step, tag instances based on the classes can be created from the OPC item list. With the defined faceplates a quick and easy visualization in Freelance Operations is available.

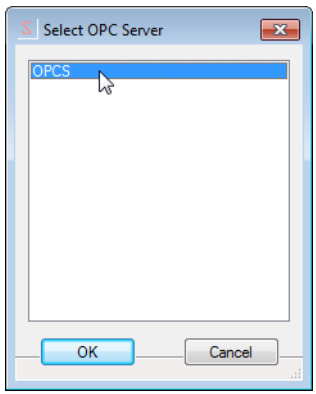
A single OPC item can be instantiated as a variable in the Freelance project.

3.1.1 Call OPC item list and browse for OPC items

The OPC item list can be opened from the **System** menu. It opens in a separate window.

> Project tree > **System** > **OPC item list**

Click **Synchronize** and select the required OPC server from the list.



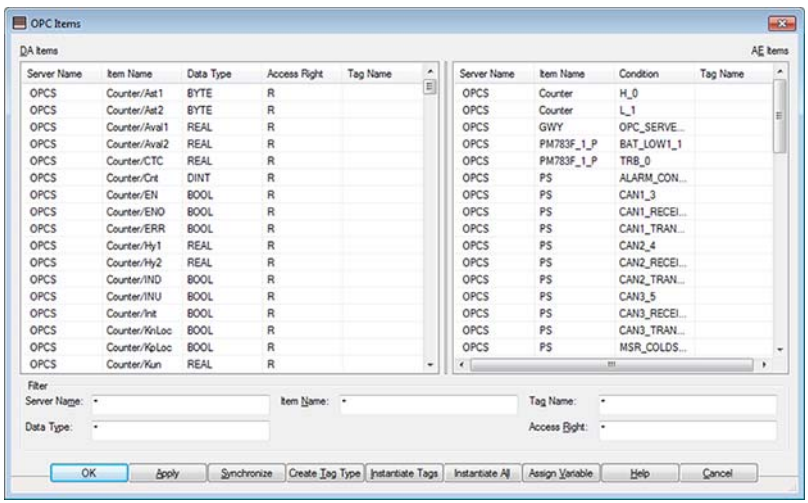
OPC_Items_02.png

After clicking **OK**, all OPC items of this OPC server which can be reached through the browser interface are added to the OPC item list dialog.

When importing OPC items from third party systems, check that the data type imported is the best match for the Freelance data types. If necessary, the user can change it to the appropriate data type manually in the OPC item list.

3.1.2 Structure of the OPC item list

The OPC item list is structured as follows:



OPC_Items_01.png

The DA items are displayed in the left pane of the **OPC items** dialog box and the AE items in the right pane.

DA Items	
Server Name	OPC server name
Item Name	OPC DA data item name
Data Type	OPC item data type, refer to Section 1, Variables
Access Right	Access rights can be of the following 3 types: **R**: Indicates a read access on the DA item **W**: Indicates a write access on the DA item **RW**: Indicates a read as well as write access on the DA item
Tag Name	Name of the Tag using the OPC DA item
AE Items	
Server Name	OPC server name
Item Name	OPC AE data item name
Condition	Type of alarms for the AE items
Tag Name	Name of the tag using the OPC AE item
Filter	The OPC item list can be filtered using masks for the columns **Server name**, **Item name**, **Tag Name**, **Data type** and **Access Rights**. When the user changes the filter mask for a column, both DA and AE item lists will be updated immediately using the filter.

3.1.3 Sort the OPC item list

The OPC item list can be sorted by using values in a specified column. Clicking the header of a specific column of one of the lists, the item list will be re-sorted immediately according to the values of the items in this column. The current sort order (down or up) for the column will be shown in the header. The sort order for this column will be toggled by next sorting.

> Click the header of one of the columns in the DA/AE list

DA/AE list is sorted using values of the column

3.1.4 Edit the OPC item list

The OPC item list is used to modify, add, export and import OPC items.

Modify the data type

> Click the cell in the **Data type** column.

> Select the data type from the drop-down list.

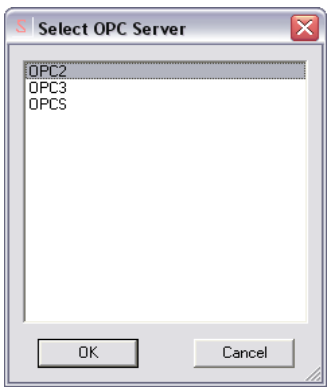
Refer to Section 1, Variables for the Data Types available.

When importing OPC items from third party systems, check that the data type imported is the best match for the Freelance data types. If necessary, the user can change it to the appropriate data type manually in the OPC item list.

Browse the OPC items from other OPC server

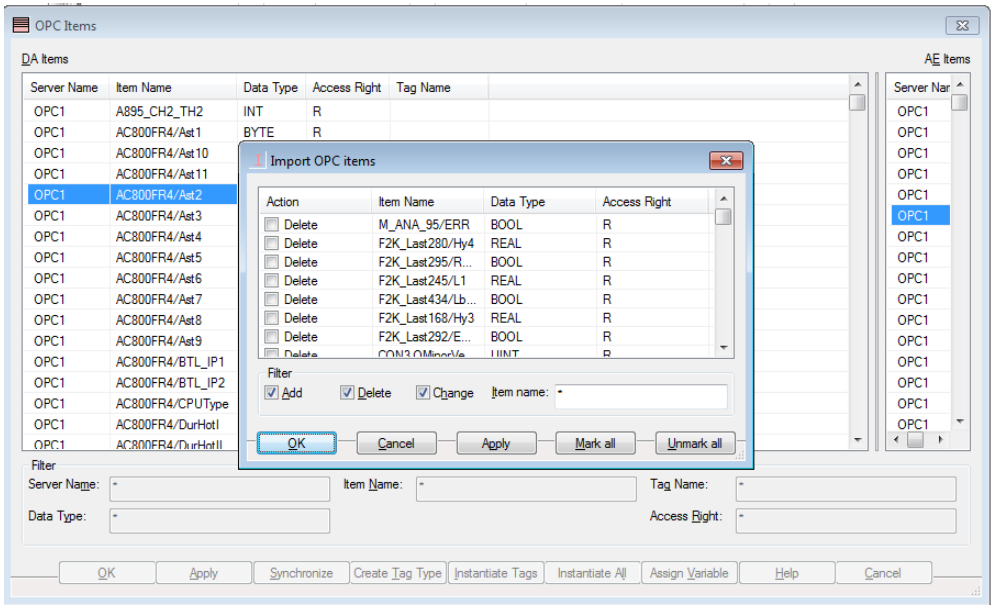
> Context menu (right-click) > **Browse**

Select a specific OPC server that will be browsed for its OPC items.



select OPC Server.bmp

Once an OPC Server is selected, the **Import OPC items** dialog box is displayed. This dialog box lists the OPC items that are available from the OPC server (i.e. the OPC items that are not already added to the OPC item list).



import OPC item dialog.png

> Tick check box to select the OPC items > **OK**

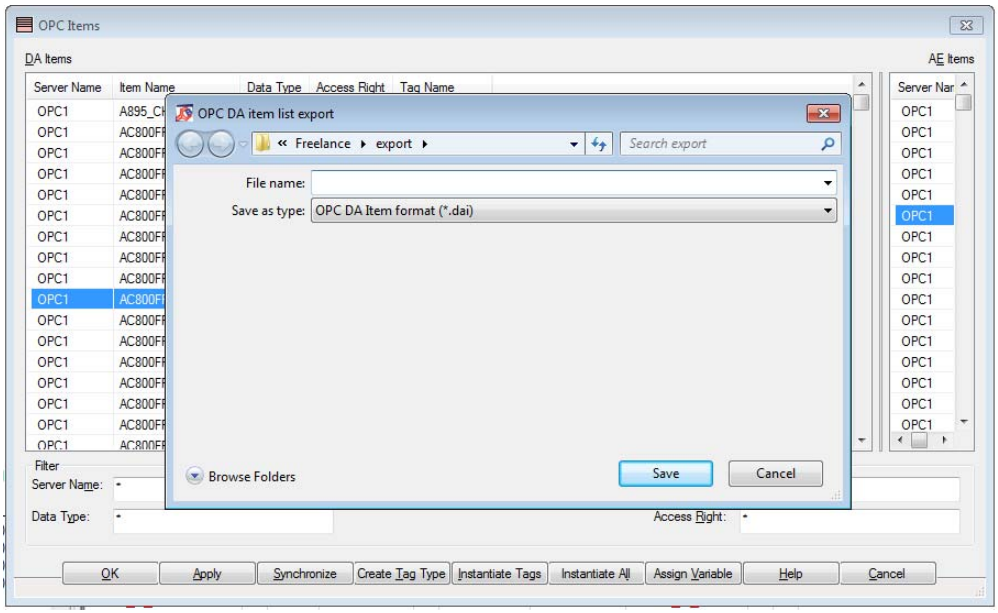
The **OPC items** dialog box is updated with the new OPC items that are selected.

When importing OPC items from third party systems, check that the data type imported is the best match for the Freelance data types. If necessary, the user can change it to the appropriate data type manually in the OPC item list.

Export OPC items

> Context menu (right-click) > **Export**

Selected OPC items in the OPC item list are written into a file of type *OPC data item format* (*.dai).



OPC DA Item list export.png

Import OPC items

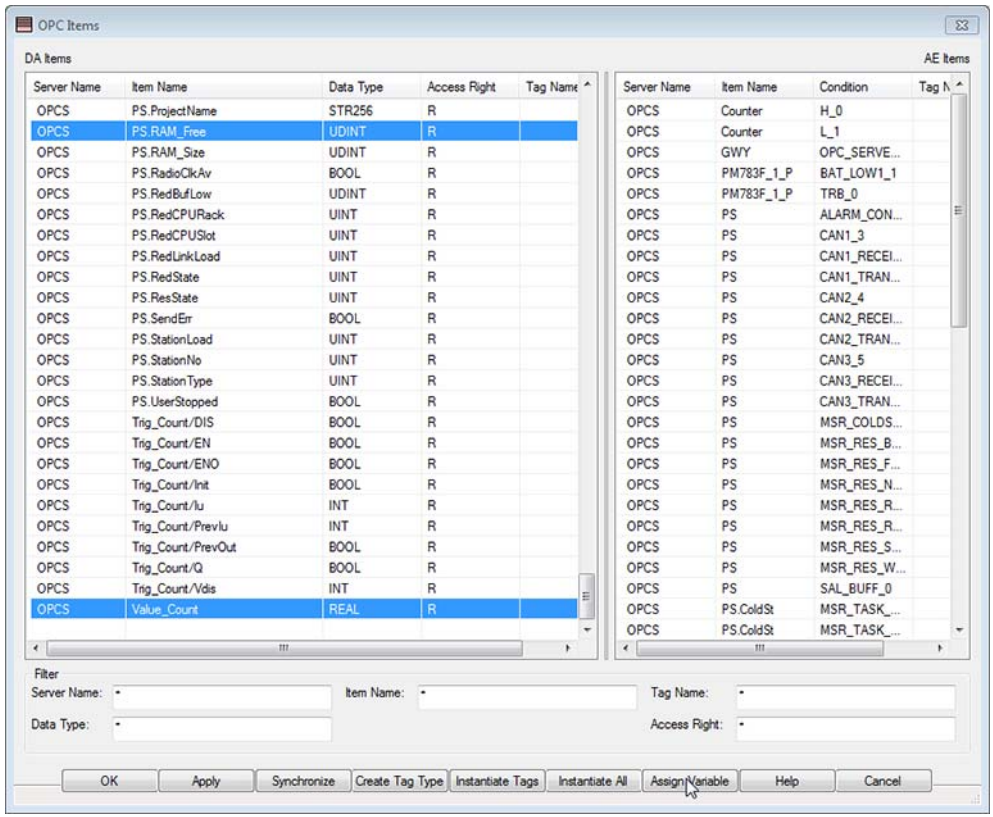
> Context menu (right-click) > **Import**

Imports the OPC item(s) from an existing DAI file.

3.2 Assign variable

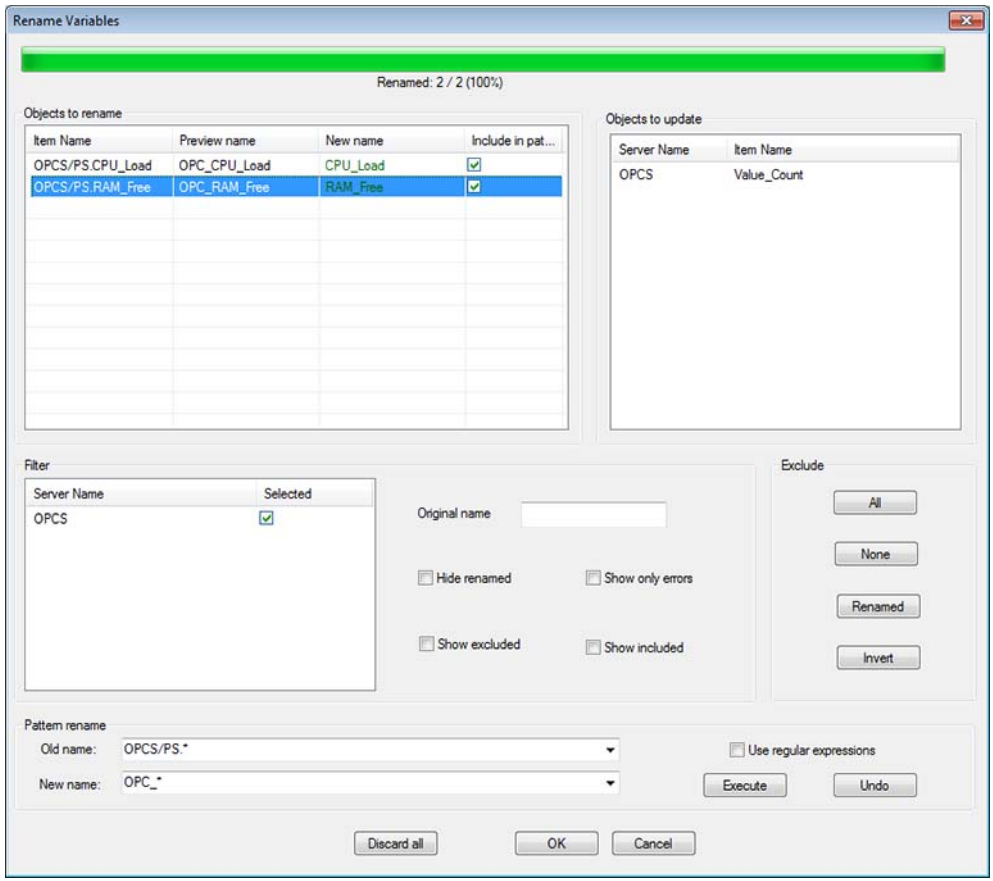
A single OPC item can be instantiated as a variable in the Freelance project. Select the OPC items from the OPC item list that are to be used in free graphics or trend displays and assign them to variables.

> Select OPC item(s) in **OPC item list** dialog > **Assign variable**



insert OPC_Items_03.png

A new dialog, called **Rename variables**, shows the OPC items and new suggested variable names. Specify the required “variable name” in the *New name* column and click on OK. All the OPC items with the new names will be assigned as a variable in the variable list.



OPC_Items_04.png

Objects to rename

These are the OPC items that are not assigned to a variable.

Item name OPC item name

Preview name Displays a preview of the new name as a result of pattern renaming, while modifying the old name of the OPC item.

New name Click into the cell to enter a new name for the OPC item.

The new name assigned to an OPC item should be unique. The characters used in the new name must meet the following conditions:

- As symbols only “_” and “§” are allowed.
- Blank spaces are not allowed.
- A variable name consisting of only integers is not allowed.

The system assigns a predefined *New name* for an OPC item by scanning the item name from right to left until a separator or a space is encountered.

Errors in the new **name column** are highlighted in RED.

If the new name is displayed in red, this means that the variable name already exists. If the cell of the new name is displayed in red, this means that the new name contains characters that are not allowed.

Include in patterning

If the Include in patterning check box is selected then the **Preview Name** gets updated in the **New name** column. The Preview name is specified by the **Rename Pattern** area (refer to Assign variable on page 90).

The **Include in patterning** check box is disabled automatically when an OPC item is manually renamed in the **New name** column.

Objects to update

These are the OPC items that are already assigned to a variable.

Server name OPC server name.

Item name OPC item name.

Filter Select to filter the Object to rename list by the OPC Server.

Original name Select to filter the **Object to rename** list by the OPC item name. This filter is similar to the windows search function.

Hide renamed Select to hide the OPC items that are already renamed.

Show only errors

Select to show only the items with errors

Show excluded Select to show the items whose names the user does not wish to change.

Show included Select to show the item names the user wants to change. By default all items are included for renaming.

Exclude

All Clears the **Include in patterning** check box for all the OPC items.

None Selects the **Include in patterning** check box for all the OPC items.

Renamed Clears the **Include in patterning** check box for all the renamed OPC items.

Invert Inverts the current state of the **Include in patterning** check box.

Pattern Rename

Renames the OPC item based on a defined pattern. This is based on the standard pattern renaming algorithm.

Old Name The name of the OPC item which is to be renamed.

New Name The new name for the OPC item.

Use regular expressions

☐ Only the wild card algorithm can be used for selecting a group of OPC items to be renamed.

✓ The pattern rename algorithm is used to select the OPC items. It is possible to group characters within an OPC item name string, and the New name can be defined based on combinations of these groups.

Execute By clicking **Execute**, New name for the OPC item will get updated by the Preview Name if the **Include in patterning** check box is enabled, under the **Objects to rename**.

Undo Click to undo the last renaming of an OPC item

To rename an OPC item: Specify if an expression will be used by selecting or clearing the **Use regular expressions** checkbox > Type an expression or a wild card value in the **Old name** field > Type the desired rename pattern in the **New name** field > **Execute**

Patterning Algorithm

Based on the OPC item name the user has to define its pattern to parse and extract the pieces from item name. This can be done with help of the symbols provided in the table below.

Symbol	Definition
()	defines a group
[]	defines a range of characters
.	every character
*	multiples of character
^	exclude a character or set of characters
\	matching character
+	repeats the previous item once more
- (hyphen)	it specifies a range of characters

The following example shows how to extract parts of an OPC item name and assemble a new variable name out of it.

Example: “OPCS/PS.CPU_Load” ==> “CPU_Load_PS_1”

In this case “PS” and “CPU_Load” has to be extracted from “OPCS/PS.CPU_Load”. “PS” falls between “/” and “.”, so that it is possible to design the pattern to have 3 groups:

1. The first group must read the OPC item name up to “/”. The pattern is “(.*)\/” which means that every character from the beginning until the “/” will be added to this group (“OPCS”).
2. The second group must continue to read the OPC item name up to “.” but excluding the preceding “/”. The pattern is “([^\/]*)\.” which means that every character without the “/” will be added to this group until the “.” is reached (“PS”).
3. The third group must continue to read the OPC item name up to the end, but excluding the preceding “.”. The pattern is“([^\.]*)” which means that every

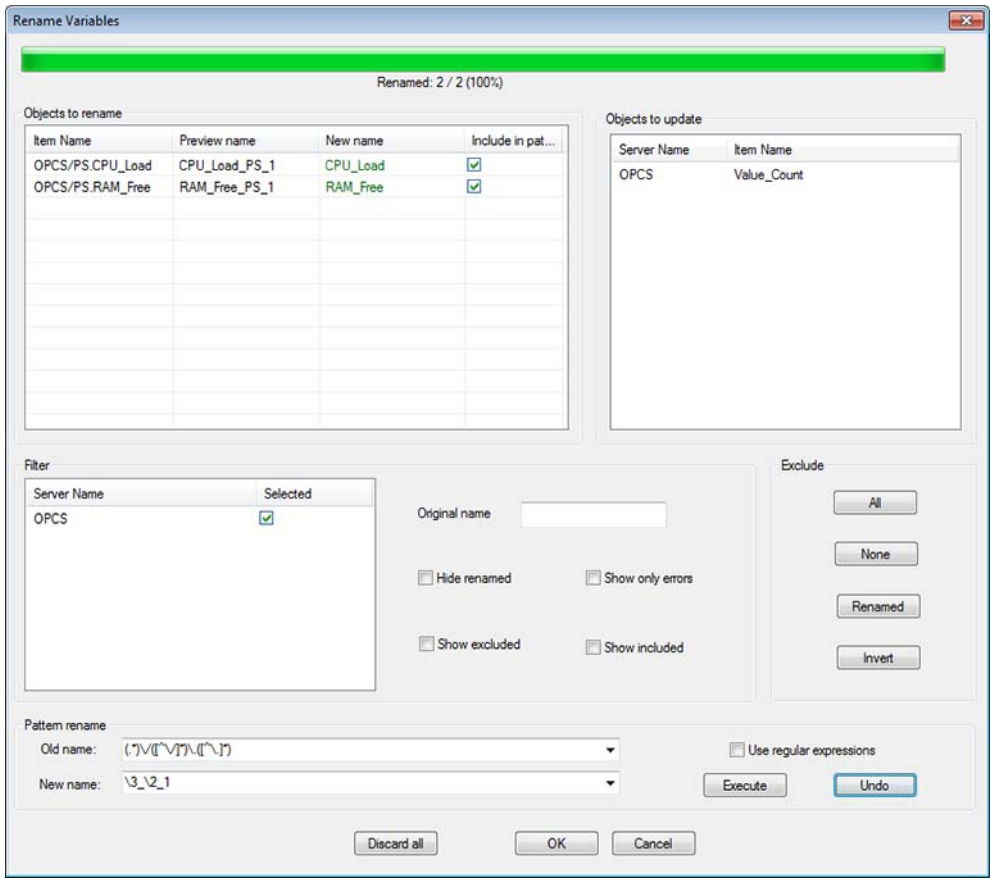
character without the “.” will be added to this group until the end is reached because it is not delimited (“CPU_Load”).

The final pattern is “(.*)\/([^\/]*)\.([^\.]*)” and must be entered into the field **Old name**.

To construct the new name “CPU_Load_PS_1”, the user can combine his above mentioned groups as follows:

1. First, the string out of group 3 is used. The pattern is “\3” which means that every character of group 3 will be added (“CPU_Load”).
2. Second, the character“_” follows. The pattern is “_” which means that this character will be added (“_”).
3. Third, the string out of group 2 follows in the new name. The pattern is “\2” which means that very character of group 2 will be added (“PS”).
4. Finally, the string“_1” follows. The pattern is “_1” which means that these characters will be added (“_1”).

The final pattern is “\3_\2_1” and must be entered into the field **New name**.



OPC_Items_05.png

Discard All The renaming of the OPC items is undone; the contents of the **New name** column in the Objects to rename list is cleared.

OK For the entries in the column **New name** a variable is generated in the variable list. For information on variables, refer to Section 1, Variables.

Cancel Close the dialog without any changes to the system.

3.3 Standard library of OPC_FB-Classes

A standard library of OPC_FB-Classes specifically developed for the Freelance system will be delivered along with the Freelance software.

Import the standard OPC_FB-CLASS library:

> Project tree > **Edit** > **Import block**

> Browse the **<Freelance_Installation_Folder> \export** folder and select one of the standard libraries available (FreelanceSampleTagType)

The imported block moves to the POOL. Drag and drop the standard OPC_FB-CLASS library under the SOFTWARE node.

3.3.1 OPC_FB-CLASS and instances

To create a tag, the user first creates the OPC_FB-CLASS. After this, the user can create instances of this OPC_FB-CLASS. There can be as many instances of an OPC_FB-CLASS as desired.

An instance is the executable form of an OPC_FB-CLASS. Different instances are identified by their tag names. Each instance works with values specific to that instance.

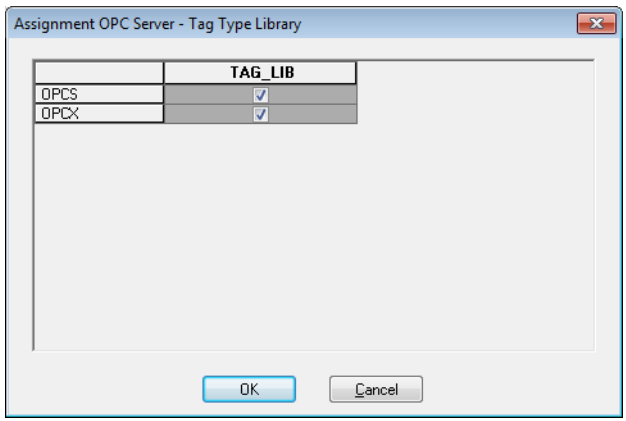
3.3.2 Create an OPC_FB-CLASS library

The OPC_FB-CLASS library (OPC_FB-LIB) must be created with a name in the project tree. Within it, an unlimited number of OPC_FB-Classes can be declared.

A maximum of 100 OPC_FB-LIB notes can be created per project.

The user can define which OPC server should have access to the OPC_FB-Lib for instantiating OPC_FB-Classes.

> Project tree > double-click the **OPC_FB-LIB** node



OPC_Items_09.png

For more information on creation of OPC_FB-LIB, refer to ***Engineering Manual System Configuration, Project tree.***

3.4 Definition of OPC_FB-CLASS

An OPC_FB-CLASS is made up of the following components

- OPC_FB-CLASS interface
- Faceplate

3.4.1OPC_FB-CLASS interface

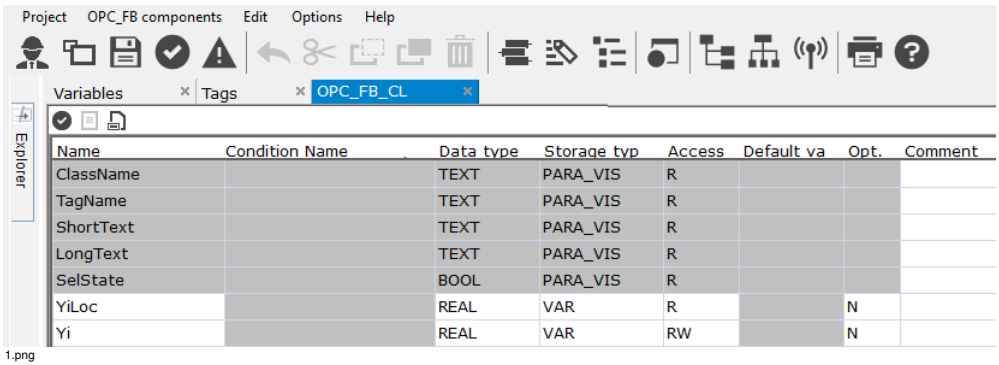
The interface of an OPC_FB-CLASS consists of a list of variables. Some standard items such as Class name and tag name exist for each class; in addition, any variables can be freely defined. All entries in the block interface can be used for the creation of a faceplate of this class. Each of the user defined variables can be marked as optional; this means an instance of this class can be created without this parameter.

Only after passing the plausibility check in the project tree, the OPC_FB-CLASS can be used within the project. The faceplate of a OPC_FB-CLASS is created in the faceplate editor. The faceplate editor offers the full functionality of the graphic editor.

Interface editor

The interface of an OPC_FB-CLASS can be created by entering data directly in the interface editor or by grouping OPC items in the OPC item list. The interface editor of a class is called from the project tree:

> Project tree > double-click the **OPC_FB-CLASS** node



1.png

If the user imports OPC items from other systems (not Freelance), it must be verified if the data format can be handled by Freelance or the user must correct them manually.

The individual entries can be selected by a double-click or by using the menus. Entries can be made directly in the **Name**, **Comment** and **Optional** fields. The **Data type, Storage type, Access** fields can only be filled in using the pop-up windows that appear.

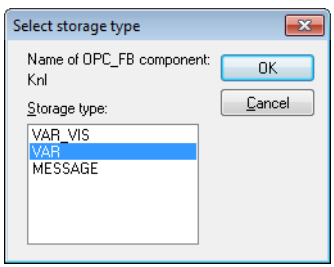
Name Freely choose variable name. Conventions for the naming of variables apply. Upper or lower case are both allowed and are differentiated. All names within the OPC_FB-CLASS must be unique.

Condition name
For alarm messages the message type is mapped via the condition entry.

Data type All Freelance basic data types are available for selection.

Storage type	The storage type determines the usage and runtime availability of the entry. See below.
Access	The Access determines how the variable is used in the OPC_FB-CLASS. The available Access types are R (read), W(write) and RW (read write).
Default value	Default value
Opt.	Y: This interface entry is optional; an instance of this class can be created without this parameter. N: This interface entry is mandatory; it is not possible to create an instance of this class without this parameter.
Comment	Any desired comment text for documentation purposes.

Storage type



OPC_Items_06.png

A storage type is assigned to each interface variable of an OPC_FB-CLASS. The storage type determines how the variable is used inside the OPC_FB-CLASS. The storage type determines where the runtime value of the variable is to be found.

The available storage types are VAR_VIS, VAR and MESSAGE storage types are used for internal processing. They do not participate in configuration.

VAR_VIS	Internal variable of Freelance Operations. VAR_VIS variables can only be read or written within Freelance Operations.
VAR	Variable; can be read and written via the OPC server.
MESSAGE	Alarm message; can be read and written (acknowledged) via the OPC server.

PARA_VIS — These entries are not available via the OPC server, but may be needed or at least useful in the project, for example the tag name and the short text. These entries can be specified during configuration of the instances and visualized in the faceplate in Freelance Operations. PARA_VIS variables cannot be changed from commissioning mode

Predefined variables

The following predefined variables are for display of general OPC_FB-CLASS data in the faceplate. Every OPC_FB-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 OPC_FB-CLASS.
TagName	TEXT	PARA_VIS	Contains the tag name of the block instance.
ShortText	TEXT	PARA_VIS	Contains the short text of the block instance.
LongText	TEXT	PARA_VIS	Contains the long text of the block instance.
SelStat	BOOL	VAR_VIS	Indicates whether the faceplate is selected. TRUE = Faceplate is selected FALSE = Faceplate is not selected

3.4.2 Modify an OPC_FB-Classes

The Freelance system provides the user the option to modify OPC_FB-Classes at any time.

There are three ways in which an OPC_FB-CLASS can be modified.

Add a variable (selector)

It is possible to add variables to already existing OPC_FB-Classes via the OPC_FB-CLASS interface.

A corresponding OPC item should be added (created) in the OPC item list.

Once a new variable is added to an OPC_FB-CLASS, all the instances associated with that OPC_FB-CLASS lose their connection to the tag name.

All instances of this block class must be created new.

Delete a variable (selector)

It is possible to delete variables from existing OPC_FB-Classes via the OPC_FB-CLASS interface.

There is no need for re-instantiation of the corresponding block instances.

Change the data type of the variable (selector)

It is possible to change the data type of the variable in the already existing OPC_FB-Classes via the OPC_FB-CLASS interface.

The data type of the corresponding OPC item should also be changed in the OPC item list.

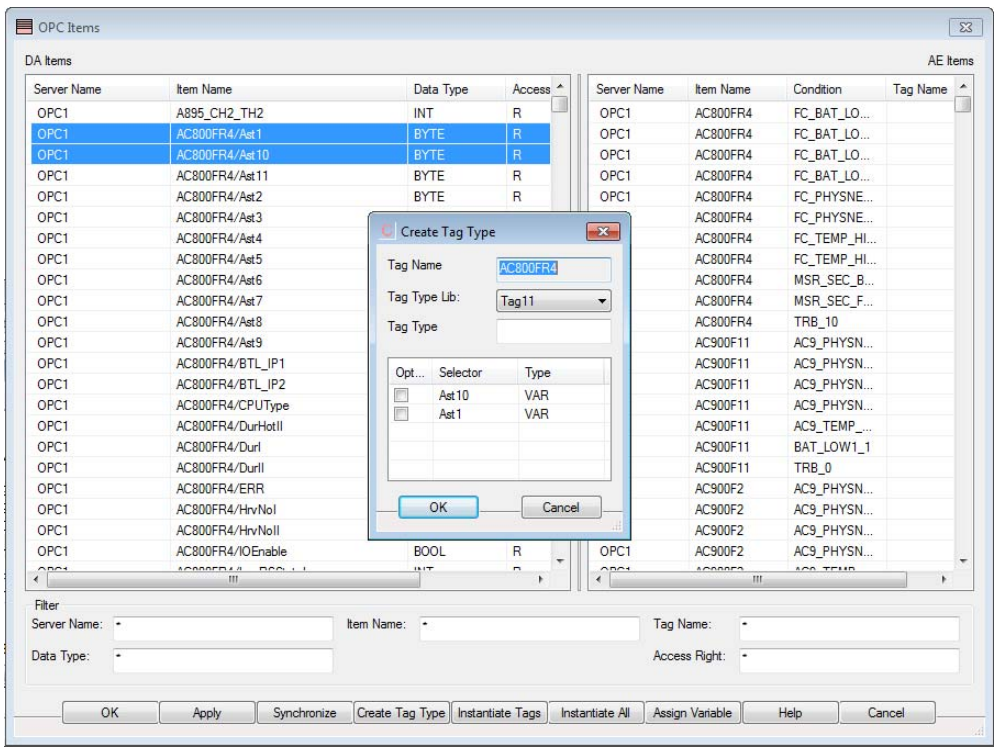
Once the data type for the variable is changed in an OPC_FB-CLASS, all the instances associated with that OPC_FB-CLASS will lose their connection to the tag name.

All instances of this block class must be created new.

3.4.3 Create an OPC_FB-CLASS

> Select the OPC item(s) in the **OPC item list** > **Create tag type**

A new OPC_FB-CLASS can be created or an existing OPC_FB-CLASS can be modified using information from OPC items selected in the Item list.



CreateTagType_new.png

The OPC items selected should differ only with respect to the selector part in their name.

Tag Name Shows a possible tag name. Item names of the selected OPC items are used to extract a common tag name by using the configured OPC server pattern.

Tag Type Lib All OPC_FB-CLASS library nodes assigned to the OPC server are available in this list box.

Tag Type Name for the new OPC_FB-CLASS

Check the tag name, select an OPC_FB-CLASS library, define a name for the OPC_FB-CLASS, check the selectors that should be optional and click **OK**

Newly created OPC_FB-CLASS are saved to database and added to the project tree.

If an existing OPC_FB-CLASS is modified and it is used by items in the list, the column “Tag Name” should be updated and a new plausibility check is needed.

If an OPC_FB-CLASS with the same structure but a different name already exists, a warning message is displayed. Click **OK** to confirm.

If an OPC_FB-CLASS with the OPC_FB-CLASS name already exists, a warning message is displayed and a new (unique name) must be assigned.

3.4.4 Faceplate for an OPC_FB-CLASS

For each function block class one faceplate can be created to display instance-specific values in Freelance Operations.

> Right click the selected **OPC_FB-CLASS** > **Insert next level**

> Select **Faceplate (FB-FPL)**

OPC_FB-CLASS faceplates are created with the faceplate editor. The faceplate editor offers the full functionality of the graphic editor.General description of faceplate editor

When a faceplate for OPC_FB-CLASS is selected in the project tree, the graphic editor is started in faceplate mode (faceplate editor).

> Project tree > Double-click the OPC_FB-CLASS faceplate (**OPC_FB-FPL**)

Creation of the faceplate graphic is nearly the same as the creation of a graphic display. The difference is in the availability of the variables that can be used for graphics animation. While in the graphics editor all global variables can be used, only the entries from the class interface can be used in the faceplate editor. For more information, see ***Engineering Manual Operator Station, Graphic display.***

3.4.5 Check OPC_FB-CLASS

The plausibility check of an OPC_FB-CLASS includes checks for the correctness of the OPC_FB-CLASS interface and the faceplates. The OPC_FB-CLASS is considered plausible only if there are no errors during the check. The plausibility check comprises of checking the interface declaration and faceplate plausibility.

3.4.6 Lock OPC_FB-CLASS

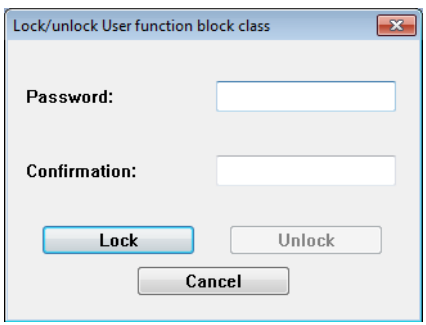
It is possible to lock the implementation of a OPC_FB-CLASS with a password.

Such locking makes it possible to hide the internal structure of the OPC_FB-CLASS (data structure) from the user, i.e. to make the OPC_FB-CLASS instances appear in their external representation only, like standardized function blocks. Similar to standardized function blocks, only the parameters are then configurable and can be commissioned.

A locked user function block cannot be modified.

> Project tree > select user function block class

> **Options > Lock/Unlock OPC_FB-CLASS**



th015us.png

For the locking operation, the password must be 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.

3.4.7 OPC_FB-CLASS comments

> Project tree > select the **OPC_FB-CLASS** node

> **Project** > **Comment**

The comment associated with the project tree junction of the OPC_FB-CLASS is displayed as help text for the OPC_FB-CLASS 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 OPC_FB-CLASS parameter dialog and displayed in a special window.

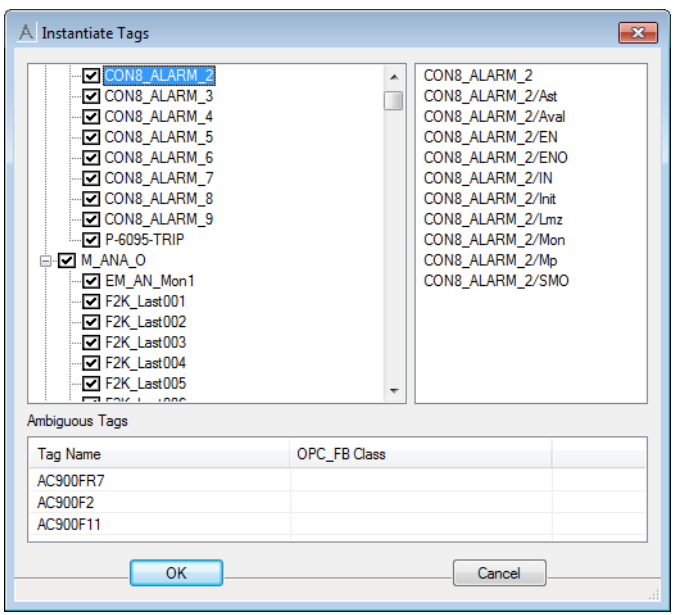
3.4.8 Export / Import

A complete OPC_FB-CLASS or its faceplate can be exported or imported.

3.5 Tag instantiation

> Select the OPC item(s) > **Instantiate Tags**

The button “Instantiate Tags” starts the instantiation proposal process for the items selected in the OPC item list.



OPCItemList_instantiate.png

Selected items are sorted in groups by finding suitable OPC_FB-Classes for them. The results are shown in the Instantiate Tags dialog box as shown above.

The tags are grouped in 2 categories:

OPCS	The selected items are categorized according to their *OPC_FB-Classes* under each OPC server.
Unknown	OPC items in the no-match list.

This dialog box has a hierarchical tree view in the top-left pane.

OPC Server /OPC_FB-CLASS/ Tags

Every tree node has a tri-state check box, indicating if the items under the node are completely, partially or not considered for the instantiation.

In the top-right side window, items contained in the selected tree node are shown.

Ambiguous tags

Tags suiting more than one OPC_FB-Class are listed along with the various OPC_FB-Classes.

Every tag in this list will be shown along with a drop-down list of the various OPC_FB-Classes that can be assigned to the tag.

The user can then select the **OPC_FB-CLASS** from the drop-down list.

Cancel

Exits from the **Instantiate Tags** dialog box without instantiating any tags.

OK

> Apply > OK

After clicking on OK the rename dialog box for tag name will be displayed. This dialog box will give the user the option of renaming tags. The OPC_FB-CLASS is instanced with the tag name. The *Tag name* columns in the OPC items dialog box will be updated after the tag instantiation.

3.5.1 Instantiate All

> **OPC items** dialog box > **Instantiate All**

The **Instantiate All** button is used to instantiate tags for all items in the lists. All the OPC items except ambiguous tags will be instantiated. The Rename dialog box for tag names is displayed. This dialog box gives the user the option of renaming tags. For more information on renaming dialog refer to **rename dialog of Assign Variable**.

When all the unique tags are instantiated and on clicking the **Instantiate All** button, a message box indicating the result of the instantiation will be displayed.

All OPC_FB-Classes that are modified (i.e. new selector added or the data type of the selector changed) have to be reinstantiated.

Instantiation of OPC_FB-Classes can be done also for variables of structured data types, if the OPC server maps tag classes are identical to structured data types. For example, a Freelance OPC server maps variables of structured data types as <variable name>.<component name> like tags as <tag name>.<component name>.

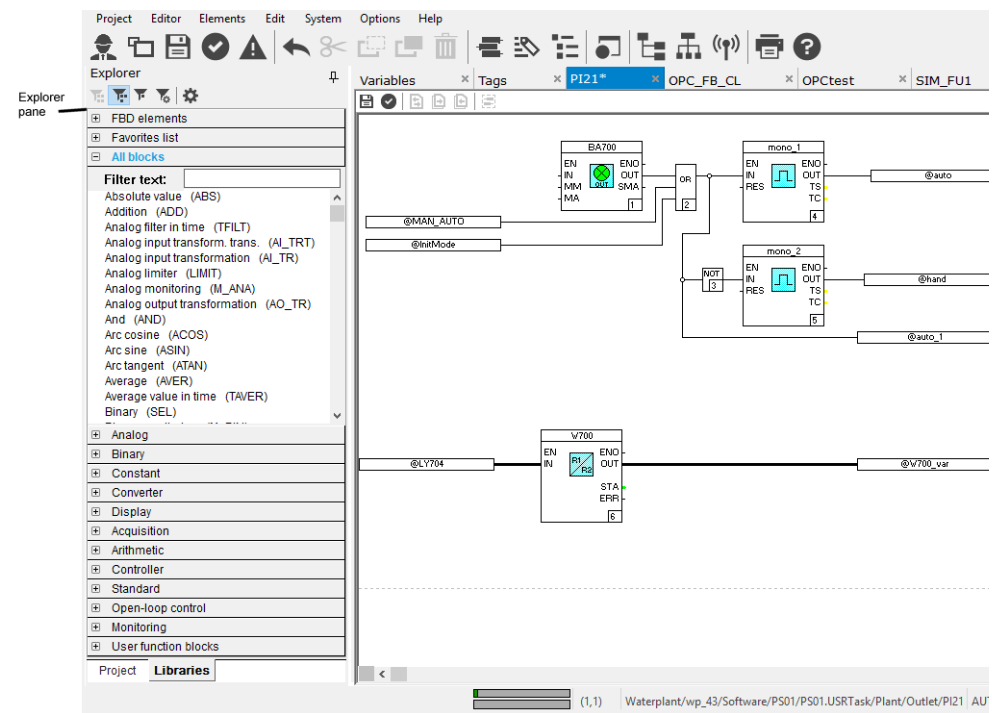
If a project contains a function block class and a structured data type with the same components, function block instances may be created during instantiation from instances of structured data types and vice versa.

This issue is addressed by renaming the components of the structured data type to make the descriptions unique. If this is not possible, ignore the unwanted entries in the renaming dialog during instantiation or delete the unwanted entries manually in the tag list after instantiation.

4 Libraries

4.1 Library – User interface

In the left pane of Freelance Configuration a list of available function block libraries can be shown instead of the project tree. During configuring of the programs thus the blocks can be accessed in a comfortable way, blocks can be selected from the list and dropped to the editor.



Libraries tab_us.png

All function blocks are grouped in libraries. The library entries can be hidden/shown depending on the filter toolbar icons. The toolbar icons are described in the following table.

Toolbar menu	Description
	**All block libraries** are shown.
	Only the **General block libraries** are shown (standard function blocks without communication function blocks, etc).
	Only the **Communication block libraries** are shown.
	**Own library list**: entries of own library list are shown.
	**Specify own library list**: a dialog opens where you can specify an own library list.

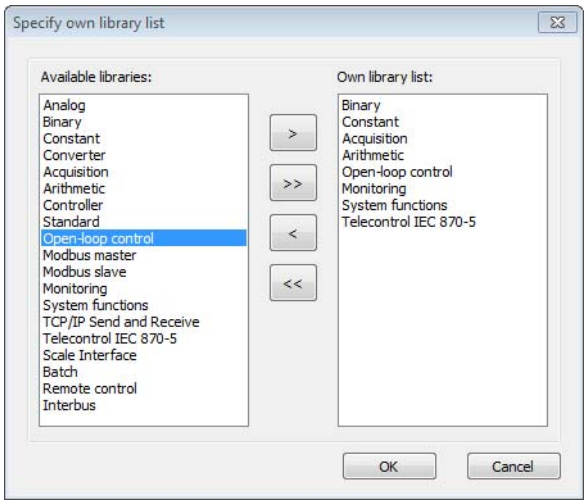
The entries **Editor specific elements**, **Favorites list**, **All blocks** and **User function blocks** are always available and cannot be hidden.

4.1.1 Specify own library list

To customize the library elements list, proceed as follows:

> Select **Specify own library list** from Library explorer toolbar

A dialog appears as shown in the figure:



Specify_own_library_list_us.png

> Select required library from **Available libraries** list

> Click [>] to add the selected function blocks library element or [>>] to add all entries of **Available function blocks** library to **Own library list**.

or

> Double-click library element to add to **Own library list** > **OK**.

The selected function blocks library element is added to the **Own library list**.

> Select required library from **Own library list**

> Click [<] to remove the selected library or [<<] to remove all the libraries from **Own library list**

or

Double-click a library in **Own library list** to remove > **OK**

The selected function blocks library element is removed from the **Own library list**.

4.1.2 Specify favorites list

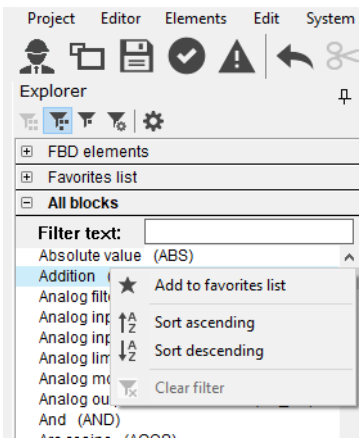
The Favorites list contains the list of favorite functions and function blocks. The entries in the Favorites list are editor-specific and can be called from the Library explorer or from the block menu. Any standard function and function block can be added to the Favorites list.

The **Favorites list** is not available for SFC programs.

To add a block to the Favorites list, proceed as follows:

> Right-click a function or function block in the list or in the editor

> **Add to favorites list**

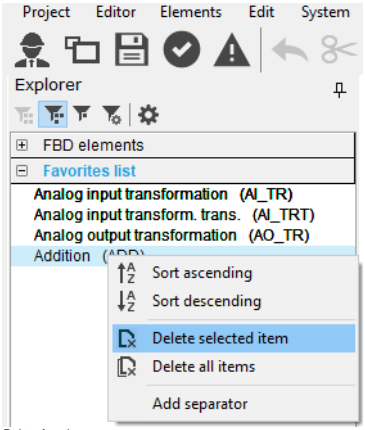


Adding to favorites_us.png

The selected item will be added to the Favorites list. If the selected item is already present in the Favorites list, it will not be added again.

To delete items from the Favorites list, proceed as follows:

> Open **Favorites list** > right-click an item > **Delete selected item**



Delete favorites_us.png

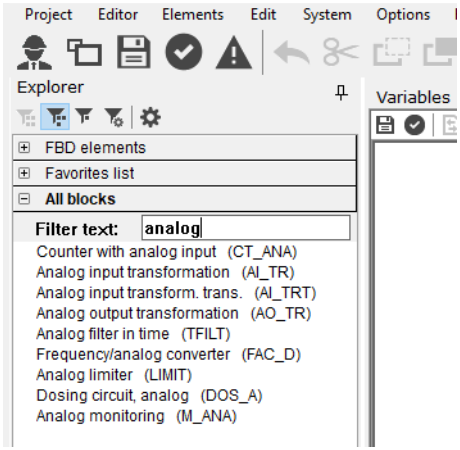
The selected item will be deleted from the Favorites list. The new Favorites list is saved.

4.1.3 All blocks

All standard function blocks are listed in the **All blocks** subgroup. This means that all blocks provided by the Freelance software are listed here, but not the classes of the user-defined function blocks and not the OPC-block classes.

Filter all blocks items

A filter option is provided in the **All blocks** subgroup to search easily for a particular function block name. The **Filter text** field appears above the items list as shown in the following figure. When entering text into the text field only those function blocks are displayed the entry of which contains the entered text.



Filter text_us.png

Clear filter list in the all block bar

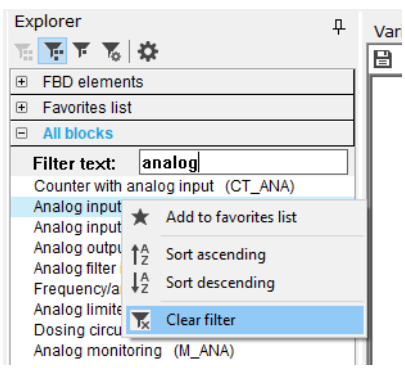
The search is not case-sensitive.

To clear the filter text and show the complete list in the **All blocks** subgroup, proceed as follows:

> Delete the text in the *Filter Text* field

or

> Right-click the list > **Clear filter**

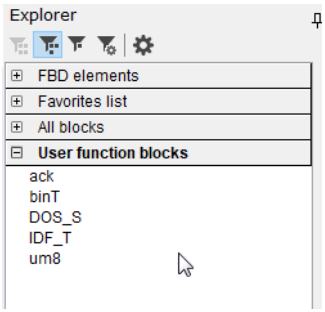


Clear filter_us.png

The **Clear filter** option will clear the filter and also the filter text from the filter text box.

4.1.4 User function blocks

The **User function blocks** subgroup items are listed in the User function block bar as shown in the following figure.



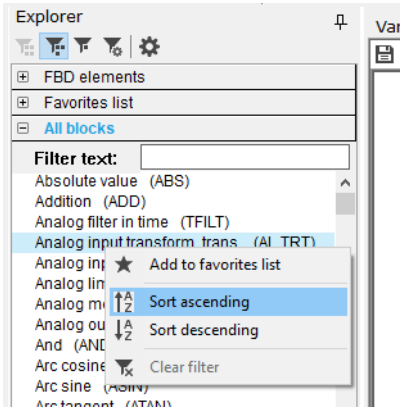
User function blocks_us.png

4.1.5 Sort elements in the list

The following subgroups provide sorting features:

- Favorites list
- All blocks
- User function blocks

The sorting feature is used to sort the list of items alphabetically in ascending or descending order.



Sort ascending_us.png

Sort ascending or descending

To sort the list of items in ascending or descending order, proceed as follows:

> Open the list > right-click the list > **Sort ascending** or **Sort descending**

4.1.6 Insert library elements into a program

FBD, LD and SFC editor

To select and drop the library elements for a non-text-based editor (FBD, LD and SFC), proceed as follows:

> Open the required subgroup in the Library explorer.

> Select an element from the list.

> Move the mouse to an active editor.

> Click the wanted target location to insert the element into the active editor.

The selected element is dropped onto the editor.

If a selected location already contains another element, the element will not be dropped onto the editor.

IL and ST editor

To select and drop the library elements for text-based editors (IL and ST), proceed as follows:

> Select the required location on the **IL/ST** editor.

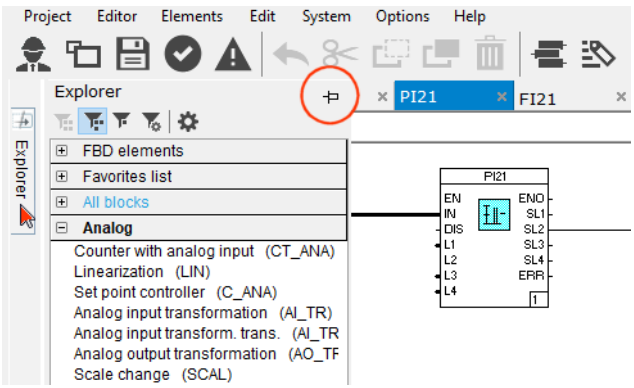
> Open the required subgroup in the Library explorer.

> Click the required element in the list.

The corresponding text of the library element is inserted in the selected location.

4.1.7 Hide and show the Library explorer

A pin button is provided on the right top corner of the Library explorer. Click the pin button to hide the library explorer to provide maximum workspace area.



Hide_libraries_us.png

When moving the cursor to the **Explorer** field at the left side of the work space, the library list is displayed temporarily and an entry can be selected.

To display the Library explorer permanently again, click the pin button in the temporary Library window.

5 Function Block Diagram (FBD)

5.1 General Description - Function Block Diagram

Function block diagram (FBD) is a graphically oriented IEC 61131-3 programming language.

The graphical functionality of the FBD permits simple positioning and connecting of functions, function blocks and their variables.

The working area of an FBD is laid out on 10x10 screen pages. The individual pages can be accessed by vertical and horizontal scrolling. The entire work area is covered by a grid. The divisions between the individual 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
- Comment fields

The signal flow of a FBD is from left to right. The signal flow lines are edited either with the mouse button and **CTRL** key pressed simultaneously or alternatively by activating an appropriate “Line drawing” mode. If the **SHIFT** key is pressed in addition to the left mouse key and the **CTRL** key, the course of the signal line is automatically determined by the system.

The named variables can be either selected from the list of system wide variables and copied in, or declared directly in the program. See Section 1, Variables.

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

As an extension of the IEC language definition, variables and their components of the structured data types may be used.

After loading the programs in commissioning mode, the editor can be activated if there is an existing connection to the process stations. The current values in the FBD program may be displayed.

For further details, refer to ***Engineering Manual System Configuration, Commissioning***.

5.1.1 Create 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” > enter a program name and optionally a short comment

Each new FBD program has a blank graphic region, the check state incorrect and the creation date as its version identifier. The name and the short comment of the program list (PL) are taken over and preset as program name and short comment of the new program; both can be changed easily.

5.1.2 Copy an FBD program

> Select program to be copied from project tree > **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**

> Enter new program name

The program is copied and assigned under a new, unambiguous name to a program list of the project. The respective configuration is copied, including the program header and program comment. 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 a version code.

5.1.3 Delete an FBD program

> Select program to be deleted from project tree > **Edit** > **Delete**

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

5.1.4 Call the FBD program editor

A program can be opened by selecting the FBD object in the Project tree. It can be opened from the Edit menu or by double-clicking the program.The FBD program is opened as a separate tab in the right pane. It can be closed using the Close button available at the right side of the opened tab.

> Project tree > Edit > Program

or

> Double-click the program

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

5.1.5 Close FBD program

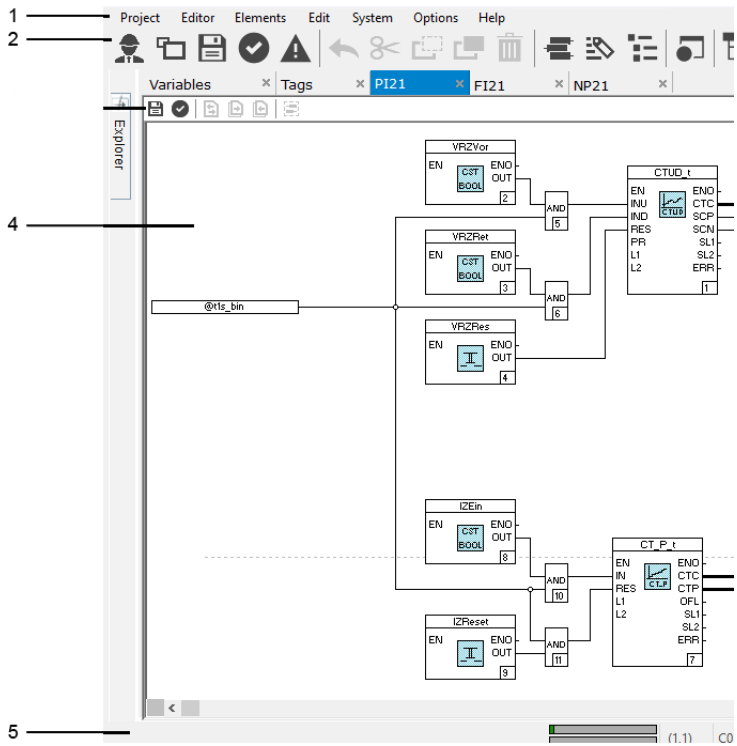
> Editor > Close

Closes the active FBD tab.

5.2 Representation of the Function Block Diagram

5.2.1 User interface of the FBD editor

The operator interface of an FBD program consists of:



FBD_Struct2_us.png

(1) Menu bar The menu entries are adapted to the active window or editor in Freelance Engineering.

(2) Common toolbar

The common toolbar is accessible from the Project Explorer and the Editor region.

(3) Editor toolbar

Frequently used commands of FBD are accessible while working in the FBD editor.

- Save editor
- Check editor
- Cross references
- Find next cross reference
- Find previous cross reference
- User FB variables (Active only for the configuration of user function blocks)

(4) Graphic region/Editor region

The function blocks and signal flow lines are configured in the graphic region of the FBD program.
The graphic region provides a grid in order to facilitate positioning of the elements, while observing minimum distances. The user can place the blocks, variables, constants, comments and signal flow lines only on the grid lines. The visibility of the grid can be switched on and off.
An FBD program can be up to 10x10 pages in size. The separate pages are delimited by dashed lines. Care should be taken not to position objects on the dashed lines, as they would be split over separate pages in the documentation.

(5) Status bar The status bar indicates the name and the page of the program which is being edited and name of the user.

5.2.2 Modify default settings

Auto Router

If the **Auto Router** function is enabled, moving one or more objects automatically adjusts the connection lines. Furthermore, the simplified line drawing mode is activated.

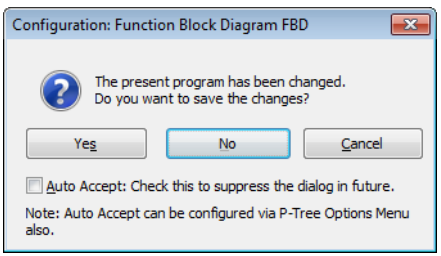
> Options > Auto Router

Auto Accept

Select **Auto Accept** to automatically save any changes in the current editor before switching to another editor.

> Options > Auto Accept
or
Double-click **AUTOACCEPT ON/OFF** in the Status bar to enable or disable the Auto accept option

If the option is not enabled, the following dialog appears for confirmation with each editor or program change:



Config_FBD_us.png

Switch the raster on and off

> Options > Raster on

All the elements in a FBD sheet are positioned within a grid. This positioning grid is made visible by this menu selection, if it was hidden and vice versa. The setting is standard for all FBD sheets in the project.

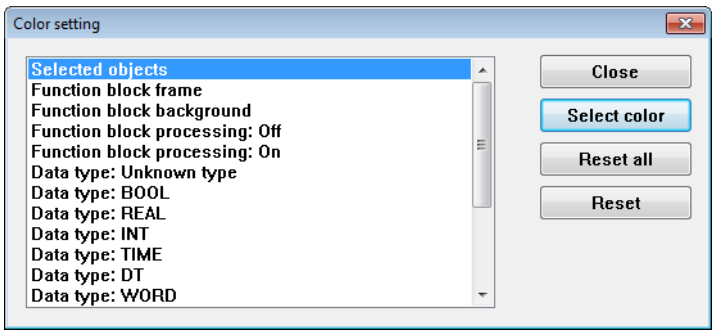
The saved settings of the last program processed are preset. The grid spacing cannot be changed.

Adjust colors

> **Options** > **Colors...**

> Select object for which the color is to be changed (e.g. function block frame)

> **Select color** > select required color



Color_settingFBD_us.png

Select color The color for the selected object can be chosen. The current color is marked.

Reset The color of the selected object returns to the default value.

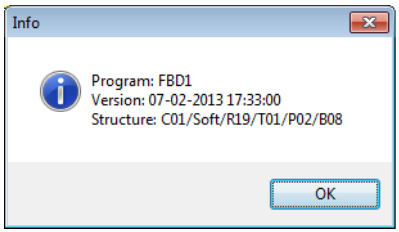
Reset all The colors of all objects are reset to the default values.

5.2.3 Display program information

Program version and position in the project structure

> **Options** > **Version...**

The program name, date of last program modification as version identification and the structure path in the project tree are shown. The structure path can be displayed in a **long** or **short format**, as set in the **Options** menu of the project tree.



di0130us.png

Program state

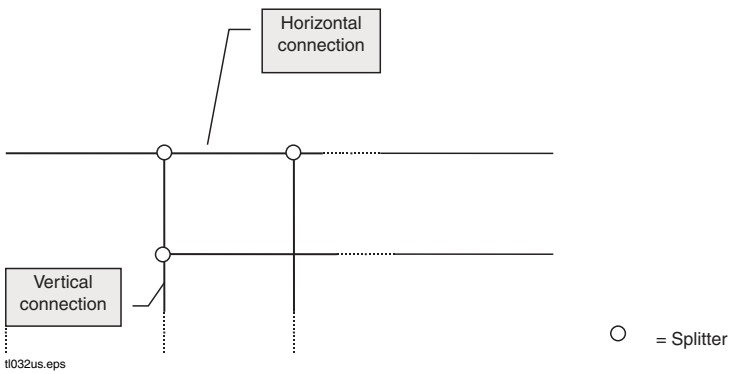
The **status bar** indicates the name and the current page of the program which is currently being edited, the position in the project tree, the current user and the license information.

Editor position (4,1)
: Shows the page (line, column) currently being edited, here the fourth page horizontally and first page vertically.

5.3 Description of FBD program elements

5.3.1 Connections and Lines

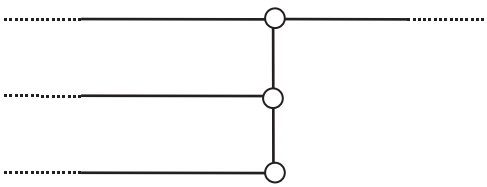
Horizontal and vertical connections can be made to variables and blocks. Connections are shown as horizontal or vertical lines. The lines are always drawn on the grid points, regardless of whether the grid is visible or not.



tl032us.eps

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.



tl033us.eps

5.3.2 Variables and Constants

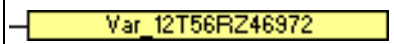
Variables and constants can be placed anywhere in the program, and are displayed and/or edited in a rectangle.

A short and a long rectangle can be selected to display the variable name and/or constants value.

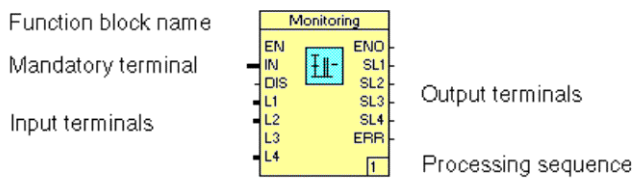
The short rectangle can display 10 characters. If the space in the rectangle is too small to display the complete label length, the overflow is indicated by '….'. The complete label is displayed as ToolTip. Alternatively the long rectangle can be selected for a permanent display of the complete label length.

Variables can be read and written either via the process image or directly. Reading or writing via the process image is indicated by @ before the variable name.

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:
 tl011.bmp	**Variable for reading**
 tl012.bmp	**Variable for writing**
 tl019.bmp	**Short version** At most 10 characters can be displayed, Overflow indication '…'
 tl020.bmp	**Long version** Max. possible label length
 tl018.bmp	Read/write via process image
 tl023.bmp	**REAL Constant**

5.3.3 Blocks



di0150us.bmp

Frame — The block frame limits the selection area of the block. The color indicates whether the block is selected or not. To change the used color, please refer to Adjust colors on page 123.

Function block name

Unlike the functions, all function blocks are displayed with a **tag name** (max. 16 characters). All the block names are included in the system-wide **tag list**. The font color used for the function block

name is used for identifying its processing 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 abbreviation 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 or specified.

Mandatory/ Optional pins (terminals)
: Mandatory pins require the supply by the signal flow line in order to enable the block to operate correctly, while this does not apply for optional pins. To distinguish the connector pins, the optional pins are shown shorter. Some optional pins disappear, if they are configured with fixed value in the parameter dialog.

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

Processing sequence
: The code on the lower right of the block defines the processing sequence within the program.

5.3.4 Comment fields

Comment field can be positioned in the FBD page. By double-clicking or selecting **Edit / Parameters** any text can be inserted. Comments do not influence the program calculation in the process station. They are used only in Freelance Engineering to describe or comment the program.

5.4 Parameterize FBD program elements

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

> Edit > Parameters

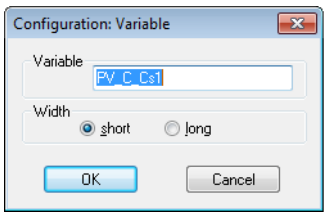
or

> Double-click the element

or

> Right-click to open context menu > **Parameters**

Configure variables



tl024us.png

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

Width

short The short version, in which only 10 characters can be displayed, will be chosen for the variable. If the variable name is longer than the display space, this is indicated by '...'.

long The long version, which can accommodate maximum-length variable names, will be chosen to display the variable.

5.4.1 Parameter definition of function blocks

Parameter types

The specifications needed for editing and displaying a block in the 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 dialog. They include information such as the block name and limit values.

Call parameter dialogs

> Select function block to be parameterized

> **Edit > Parameters**

or

> Double-click function block

The first parameter dialog of the function block is opened. All other selected elements are automatically deselected. After return from the parameter dialog, the function block is represented accordingly with the modified parameters.

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 dialog. In all cases, this applies at least to the **block name** (max. 16 characters) of a function block.

All block names entered for function blocks are summarized system wide in the **tag list**. For a description see Section 2, Tags.

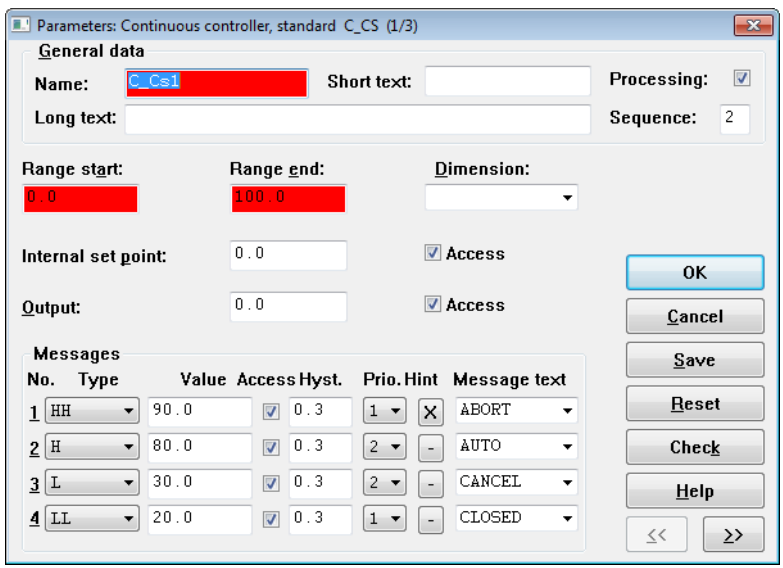
Alternative input possibility for the block name:

> Select text field Name > **F2** key

> Select block name from tag list. (Only the names defined in the tag but not yet used are available for selection.)

Handling the parameter dialogs

By virtue of the different parameters governing the various function blocks, there is no uniform parameter dialog. However, certain sections are used similarly in all or in some parameter dialogs. Besides, there are several parameter dialogs for large blocks and they can be edited in any order desired.
Using the parameter dialogs of the function block “Continuous ratio controller C_CR” the basic features are outlined below:



di0627us.png

Header line — Name and short designation of the block; if necessary number of the parameter dialog currently in use.

General data — These fields are available in all function blocks; Name, short and long text to describe the function block and also the processing status and processing sequence.

See also ***Engineering Reference Manual Functions and Function Blocks***.

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

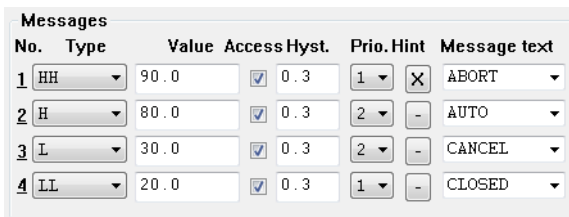
Text field — For entering block name and long text, for example.

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

The optional parameters, *Short text* 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 pin. Conversely, the pin disappears from the block display if a parameter has been entered. Consult the block description for the parameters to which this applies.

List



di0165us.png

There are lists where only entries from the preset list can be selected, eg. lists of message types and priorities. The desired entry is taken over by clicking the input field.

Some lists have an input field that can be freely edited, e.g. the message text. Either an entry can be selected from the list or a particular text entered using the keyboard.

To enhance transparency, it is recommended that short and long texts be entered for the blocks. The parameters of a block are generally preset in such a way that the block can be used for a standard application without further processing. For a description of the block parameters, refer to ***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.

5.4.2 Parameterize comment fields

Enter any free text to describe the program or any special functionality. Comments fields can be resized with the mouse and positioned to any free region in the graphic region.

5.4.3 Change the processing sequence of the blocks

> Select block > **Edit** > **Processing sequence**

> Enter processing number in block (the previous one is marked for overwrite)

or

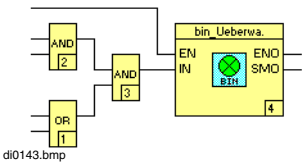
> Modify processing sequence in the parameter dialog of the function block

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 is entered that exceeds the total number of blocks used in the program, the edited block is given the total number as its processing number.

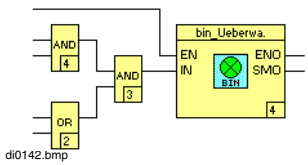
The processing number is assigned automatically in the chronological order in which the blocks are positioned.

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

In the example on the right, the modules cannot be directly calculated in the defined sequence. Variables internal to the system are used to save intermediate data. If the configuration of such a program is modified, this modification may be loaded on the controller because with each program modification, all internal variables are initialized with the value 0.

5.4.4 Define favorites list

Functions and function blocks that are required frequently for configuration can be grouped together in a separate list and block menu for easy access.

> Options > Define favorites list

For details refer to Specify favorites list on page 110.

5.5 Edit an FBD Program

5.5.1Draw signal flow lines

Signal flow lines can either be drawn explicitly or created automatically by the system. To draw the lines explicitly, horizontal and vertical “line sections” are defined; if the signal paths are to be determined automatically it is necessary only to specify the start and end points of the signal flow.

Explicit draw of 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

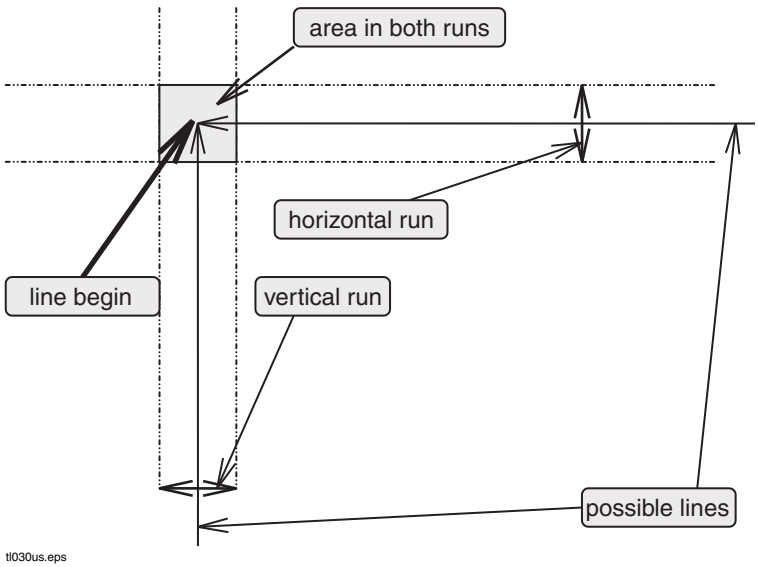
> Right-click (context menu) > **Draw lines**

The mouse pointer will change to a cross.

A single click determines the beginning of the line. When the mouse is moved either a horizontal or vertical line is drawn if the cursor is at the start of the line (within the snap) and does not cut across a block.

Every additional click terminates the current line and simultaneously defines 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.

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



Draw line

> Right-click with the mouse to define the start and end of line.

or

> Simultaneously press the **CTRL** key and left mouse button to directly draw a signal flow line.

When the left mouse button is released, this defines a horizontal or vertical line section. Releasing the **CTRL** key has the effect of exiting line-drawing mode.

Deactivate line-draw mode

> Right-click with the mouse

or

> ESC key

Automatic drawing of signal flow lines

Start of a signal flow line:
> Click the pin available in the FBD element > drag the mouse to draw the line.

End of a signal flow line:
> Release the mouse button

Auto Router enabled:

In order to draw signal flow lines automatically, click the FBD element pin. This will automatically change to the auto connect mode and will draw the connection line until you release the mouse button again. If you click any other part of the FBD element, the element is selected for other editing options.

Auto Router disabled:

Pressing the **CTRL** and **SHIFT** keys allows for automatic drawing of lines from every point in the FBD. When the left mouse button is pressed, the start point of the

signal flow line is defined. The end point is defined where the left mouse button is released.

> Click once to determine the start of the line.

When the start point has been defined, the cursor is moved with the mouse button. The possible path of the signal flow line from the start point to the current cursor position is indicated. When the mouse button is released, the signal flow line is finally defined.

Move element with/without signal connection

Auto Router enabled:

When the elements are moved in the FBD/LD editor, the signal connections are retained.

> Select FBD element (Variable/Block) > move element in the editor

Auto Router disabled:

When the elements are moved in the FBD/LD editor, the signal connections are removed.

To move the elements without removing the signal connections, execute the following steps.

> Select object >Start moving it with the mouse > press and hold the **CTRL** + **SHIFT** keys while moving the object

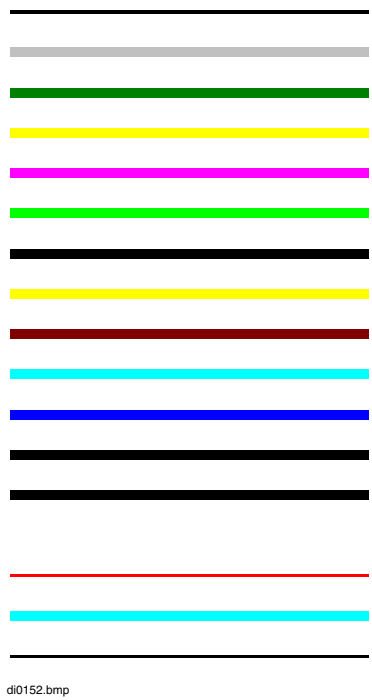
If there is not sufficient free space available in the drawing area, the signal flow line will not be drawn.

Display signal flow lines

The signal flow lines indicate the data type transported. **Selected** and **incorrect** signal flow lines are displayed by different colors.

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 Adjust colors on page 123).

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

Data type/ Processing state	Color	Display	Example
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

5.5.2 Insert FBD elements

Variables, blocks and comments can be inserted from the Library explorer or from the main menu.

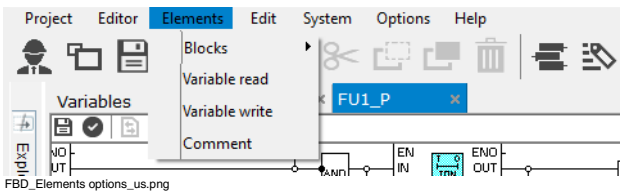
> **Explorer** pane > **Libraries** tab > Select element to be inserted

or

> Elements > Blocks/Variable read/ Variable write/ Comment

> Select element to be inserted

For more information on the Explorer pane, refer to Section 4, Libraries.



FBD_Elements options_us.png

After the element to be inserted has been selected, the cursor takes on the shape of the selected element. The element can then be positioned in the active tab in the workspace pane by left-clicking with the mouse. If the element should not fit in, the cursor reassumes its normal shape. This clears the selection of the element from the clipboard memory.

If the placing was performed successfully, the insertion operation is ended automatically.

Insert variables

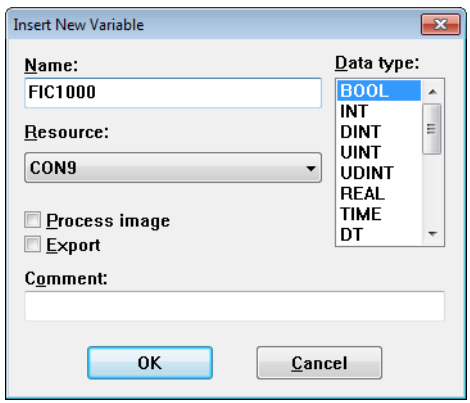
Since variables can be placed anywhere in the program, it is essential to specify whether they are to be used for reading or writing. Depending on whether a variable or constant is to be used for reading or writing, the surrounding rectangle is provided with either an input or output pin of the appropriate data type. As long as the variable is not connected with a line, access can be switched between **read** and **write** mode via the shortcut menu with **Toggle read access**.

After inserting a variable element, the name of a variable must be entered. You can enter the name of a variable already known in the project or a new name.

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

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

Once a new name has been entered, the dialog box described in the previous section for declaring a variable is displayed automatically.



di0133uk.png

Data type Identify the data type of the newly defined variable. The standard data types and all user-defined data types can be selected from the list.

Resource Sets the allocation of variables to resources. Each variable must be allocated to exactly one resource. Access to this variable by other resources is read-only.

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***.

Export ☑ Variable is enabled for reading by other resources.

Comment Any desired text can be added to a variable for clarification.

Once the variables have been defined, this setting is automatically adopted in the system-wide variable list and can be used in other programs. See Section 1, 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.

An I/O component can only be exported via a variable, not directly. This means it cannot be read in other resources by using the component name.

Select and position blocks in the program

> **Elements** > **Blocks** > choose the wanted block type.

> Move with the mouse to the desired location in the graphic region.

> 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; click to confirm).

> Position another block of the same type

or

> End positioning at any time with **ESC** key or right mouse button.

Once a block has been selected, it can be inserted in the graphic region. While being positioned, it is displayed schematically. After positioning is complete, 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 have been placed, their size can be changed immediately. 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 dialog 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.

5.5.3 Change number of inputs

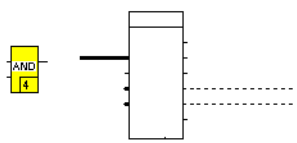
> Select block > **Edit** > Change number of inputs
> Move mouse up or down until the required number of inputs is displayed
> Confirm with the left mouse button
> End positioning at any time with **ESC** key or right mouse button.

or

> Double-click the lower boundary line of the block
> Move mouse up or down until the required number of inputs is displayed
> Confirm with the left mouse button.

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.



di0140.png

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.

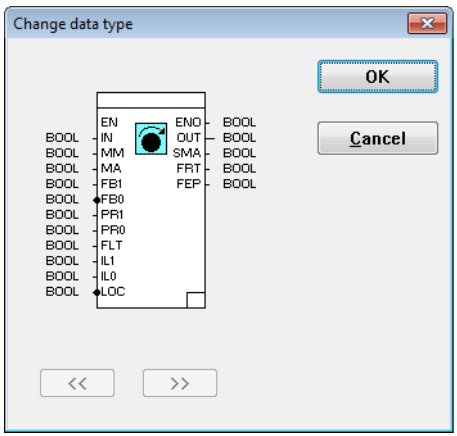
For assignment of inputs to the blocks (analog, binary, etc.) see ***Engineering Reference Manual Functions and Function Blocks***.

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.

5.5.4 Display and change data types

> Select block > **Edit** > Change data type

> Set and enter the required data type with >> and <<.



di0131us.png

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

Consult the respective block descriptions for the data types possible for each block. For more information, refer to ***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.

5.5.5 Invert a block terminal

> Press and hold **Ctrl** > left-click the block terminal to be inverted

The values of the Boolean inputs and outputs of a function block can be inverted at the block. 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.

di0151_us.png

Block with negated terminal

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

5.5.6 Change variables

> Double-click the variable to be changed > Change variable name > **Enter**

> Define new variable in the “Insert new variable” dialog.

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

or

> Double-click the variable to be changed > **F2** key > Select one of the variables already existing in the project in the “Select Variable/Component” window.

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.

Unused variables remain in the variable list and must be explicitly deleted there.

5.5.7 Cross references

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

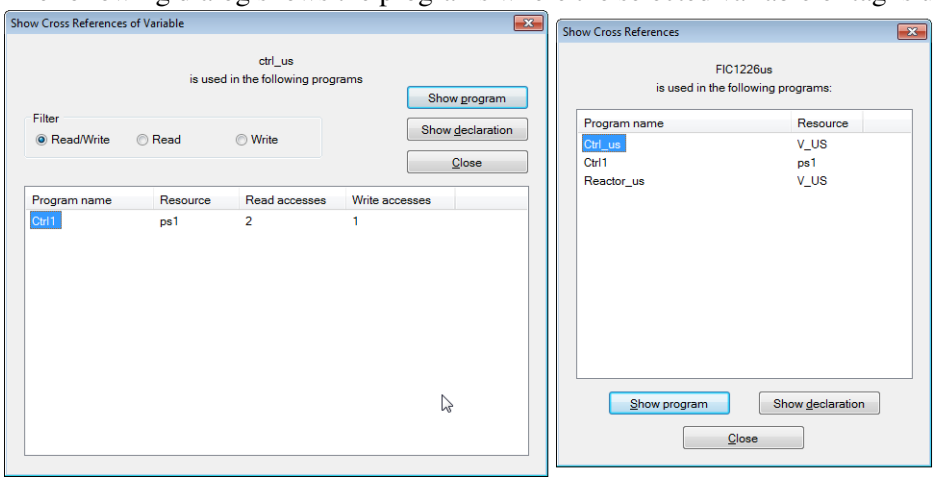
> Select a variable, I/O component or tag

> **Edit > Cross references**

or

> **F5** key

The following dialog shows the programs where the selected variable or tag is used.



CrossRef_us.png

In contrast to the variables, no read or write access is defined for the tags.

Show program

For a variable:
Call a program with prior selection of these variables, or call the module with prior selection of the I/O component.
For a tag:
Call the program with prior selection of this tag, or call the module in the hardware structure.

Show Declaration

For a tag, the tag list is called, for a variable the variable list is called. If an I/O component is used directly in the program, the I/O editor of this component is opened.

Filter A filter enables only those variables to be displayed for which read-only access or write-only access exists in the programs concerned.

After activation it is possible to branch to the programs listed as cross references.

Show next / previous cross reference

> Select a variable > **Edit** > **Cross references** > **Find next** or **Find previous**

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

5.5.8 Insert or delete columns and rows

In the current program, parts of the configuration can be “pushed apart” by insertion of columns or rows or “pushed together” by deleting columns or rows.

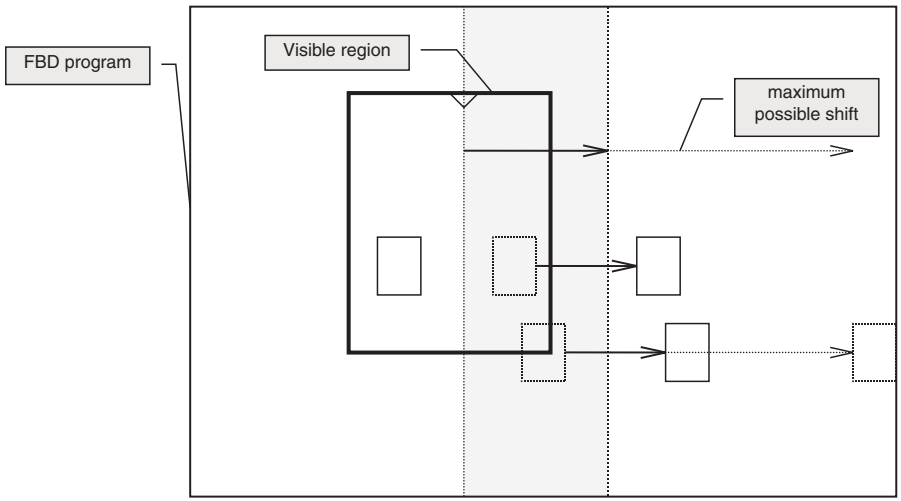
If the cursor is moved at the edge of the drawing area, a small double arrow is superimposed. If the double arrow is in black, it is possible to insert or delete columns or rows at this point. A red arrow means that insertion or deletion is not possible.

Insert or delete columns

By clicking with the mouse on the upper or lower network edge on a black double arrow, a vertical dotted line with two pointed triangles is superimposed at the edge of the drawing area. This line can be shifted to the right or left by depressing the left mouse button.

With each shift to the right by one raster unit, a column is inserted into the drawing area and the part of the configuration to the right of the line shifted to the right by one raster unit.

With each shift to the left by one raster unit, a column is deleted from the drawing area and the part of the configuration to the right of the line shifted to the left by one raster unit.



When columns are inserted or deleted, horizontal connections are extended or reduced accordingly.

Insert or delete rows

The insertion of rows corresponds to the insertion of columns. The movement markings run in a horizontal orientation. When rows are inserted or deleted, vertical lines are extended or reduced accordingly.

5.5.9 Block operations

Select program elements

Select **individual program elements**.

> Click the required program element to select

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

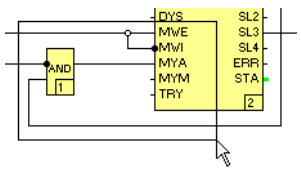
The non-selected state is preset.

Inversions and link points of signal flow lines are never displayed as selected.

Select **several program elements concurrently**

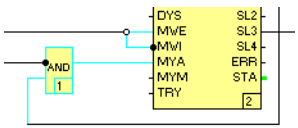
> Press and hold left mouse button > with mouse button pressed, draw a frame around the elements to be selected

All elements fully enclosed in the frame are **selected concurrently** and shown accordingly. In the case of the signal flow lines, this applies to 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**.



Place a frame

di0147.bmp



Selected program elements

di0148.bmp

Select **additional program elements**

Press and hold **SHIFT** > 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

Deselect program elements

Deselect all selected program elements

> Click on a free point in the graphic region.

or

> Select a non-selected element.

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 and hold **SHIFT** > click element to be deselected

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

Copy

> Edit > Copy

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.

Content of one editor can be copied and pasted to another editor of the same type (For example: FBD editor content can only be pasted in into FBD editor, not into IL or ST). This will enable user to modify the programs easily.

When function blocks are copied, the parameter data remain unchanged. However, the tag name is deleted in the copy, as it must be unique.

Cut / Delete

> Edit > Cut or **Delete**

If the selected elements have been cut, they can then be re-inserted in the program using **Paste**. **Cut** has the effect of overwriting any elements held in the internal storage at the time.

If elements are deleted, they can only be pasted directly after this, using **Undo**. They cannot be pasted at a later time. 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.

Paste

The **Paste** command can be used to insert elements that were previously copied or cut:

> **Edit** > **Paste**

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.

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

or

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, then a warning message is displayed and the surrounding rectangle remains active.

The selected elements are moved to a new position, while the element contours remain visible. When moving block(s), the signal connection lines will stay connected except when **Auto Router** is disabled.

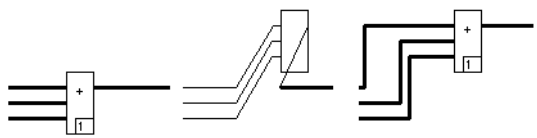
Move block with existing links

If the existing links are to be retained when a block is moved, proceed as follows:

Auto Router enabled:
> Click a selected element and drag the element to destination.

Auto Router disabled:
Select the object, then press the left mouse button. Press **CTRL** + **SHIFT** and drag the object to destination.

To move a block without existing links:
Deactivate **Auto Router**, click a selected element and drag the element to destination. The block will be moved without existing links.



di0141.bmp

Representation of a function block before, during and after being moved with existing links.

Import block

> **Edit** > **Import block**

An “Import FBD block” 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.

If the block imported contains variables that are not yet included in the variable list they are displayed in red. Selecting these variables enables their definition in the current project.

Export block

> Edit > Export block

The selected elements of a current FBD sheet can be exported to a file. An “Export FBD blocks” dialog box appears, containing a list of all previously exported files in the most recently selected export directory.

Tag names of the selected blocks will not be exported.

5.5.10 Undo an action

> Edit > Undo

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

5.5.11 Program administration functions

Save the program

> Project > Save Tab

The program is saved without exiting. Programs that are not correct can also be saved and then completed at any time.

If the project is not saved in the project tree on closing or before, changes made to the program are ineffective.

Document the program

> **Project > Documentation**

The editor for documentation is opened as a separate tab in the right pane. It can be closed using the Close button available at the right side of the opened tab, and reopened later from the main menu.

Documentation administration is opened. This is where user specific project documentation is defined and output. For a description, see ***Engineering Manual System Configuration, Documentation***.

Program header

> **Project > Header**

A program-specific short comment can be added to the program documentation header, or this can be edited.

For drawing a header/footer, see ***Engineering Manual System Configuration, Documentation***.

Edit program comment

> **Project > Comment**

A longer program-specific comment can be edited here to describe the functionality. For a description, see ***Engineering Manual System Configuration, Project manager***.

Print

> **Options > Print**

The contents of the screen are output to the standard printer.

Plausibility check

> Editor > Check

All inputs relevant to operation are checked for syntactical and contextual correctness. Errors, warnings and notes that are found are displayed in an error list. If the plausibility check detects errors, the processing state of the program is **implausible**.

The processing state of program elements that are newly entered, copied or moved is **implausible**.

This plausibility check reviews the accuracy and consistency of the program itself. To test the correctness in the project context call plausibility from the project tree. See Engineering Manual System Configuration, Project Tree, plausibility.

Error list

> Editor > Show error list

Any errors present in the program is displayed in the error list. Double-click a check message to jump to the line in the program that caused this error.

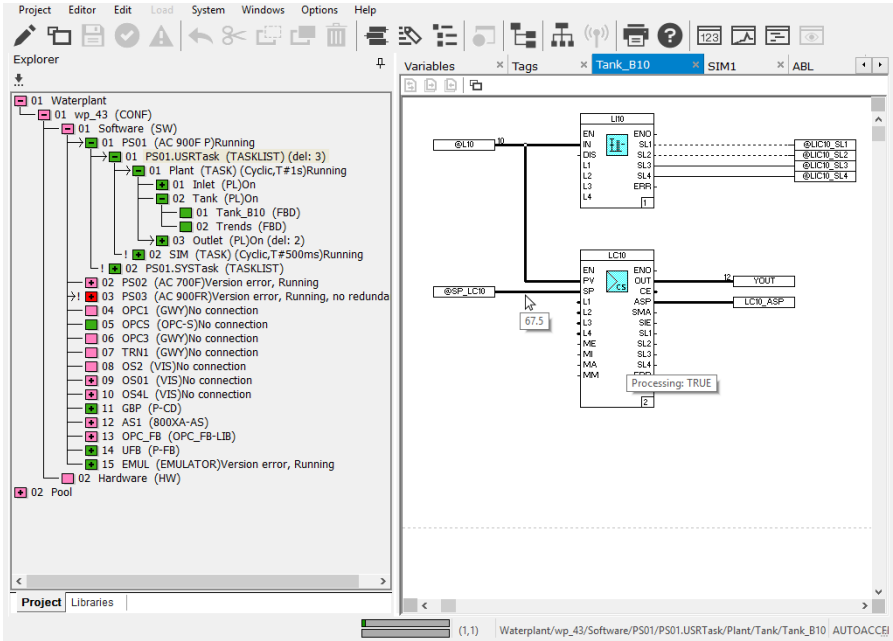
See also ***Engineering Manual System Configuration, Project tree***.

5.6 Commissioning the Function block diagram (FBD)

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

The user can go to commissioning mode directly from the editor.

If the program editor is opened in commissioning mode, that for showing the live value, CPU load can raise approximately up to by 15%.



FBD_Comm_us.png

Individual function blocks can be selected and parameters set for them. Operating modes can also be called up and modified from commissioning mode.

In addition, in commissioning mode certain program test functions are available to test the configuration.

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

logical 1 ——————————TRUE

logical 0 ---------- FALSE

If the cursor is moved over an element of the FBD program, a variable, a pin or a connection line, the current calculated values are displayed. For function blocks the current processing state is displayed.

Furthermore, variable values and input pins of the function blocks can be written once within a cycle

Writing a value to an unconnected input pin works as the configuration of a constant value for the input, but it is not visualized at the function block. This can be difficult to notice later and should therefore be used with caution.

> Right-click a variable or function block pin

> **Write values** > Enter new value > **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.

6 Instruction List (IL)

6.1 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 Freelance Engineering 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 dialogs as in the other programming languages.

The functional scope of instruction list (IL) – in contrast to the function block diagram (FBD) – is extended by jumps and loops that are called by the corresponding operators and the entry address (label) be terminated.

The signal flow is not as easy to follow or document as in FBD; therefore a comment can be edited 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.

As an extension of the IEC language definition, variables and their components of the structured data types may be used.

6.1.1 Create an 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 Section 9, Sequential Function Chart (SFC)

An IL program is created in the project tree:

> **Project tree** > select insert position in project tree

> **Edit** > **Insert above**, **Insert below** or **Insert next level**

> Select IL program from “Object selection”

> Enter a program name and optionally a short comment

Each new IL program has a blank instruction list, the check state incorrect and the creation date as the version identifier.
The name and the short comment of the program list (PL) are taken over and preset as the program name and short comment of the new program; both can be changed easily.

6.1.2 Copy an IL program

> Select program to be copied from project tree > **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**

> Enter new program name

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

6.1.3 Delete an IL program

> Select program to be deleted from project tree > **Edit** > **Delete**

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

6.1.4 Call the IL program editor

A program can be opened by selecting the IL object in the project tree. This can be opened from the **Edit** menu or by double-clicking the program. The IL Program will open as separate tab in the right pane. This can be closed from the close button available at the right side of the opened tab.

> Project tree > **Edit** > **Program**

or

> Double-click the program

The program is displayed with its current content (instructions) and can be modified.

6.1.5 Close IL program

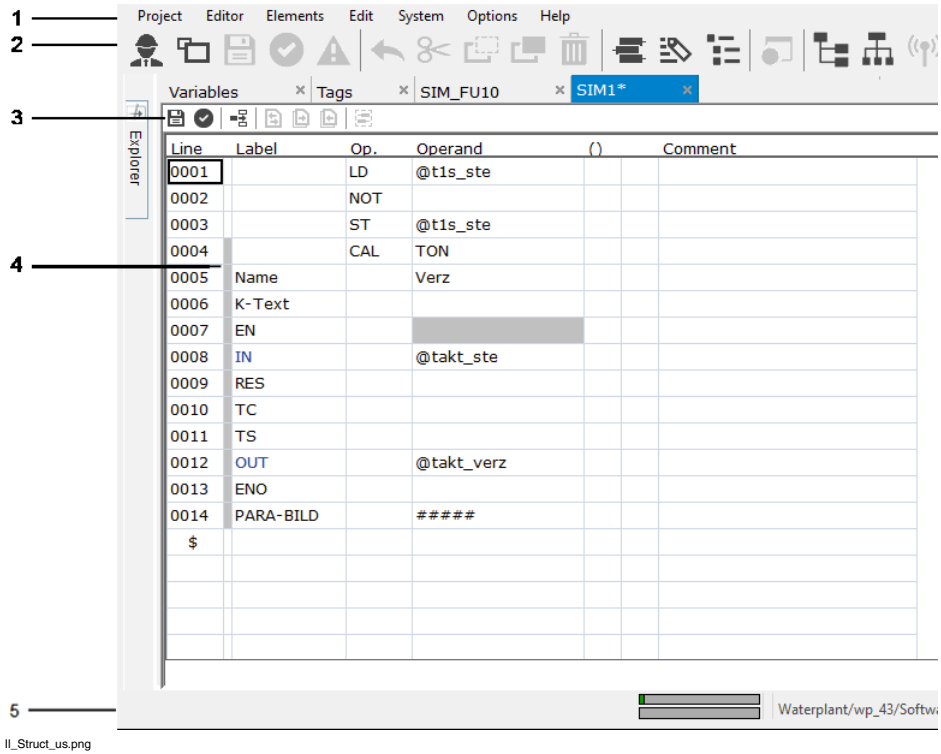
> Editor > **Close**

Closes the active IL tab.

6.2 Representation of the Instruction List

6.2.1 User interface of IL editor

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



ll_Struct_us.png

(1) Menu bar The menu entries are adapted to the active window or editor in Freelance Engineering.

(2) Common toolbar

The common toolbar is accessible from the Project Explorer and the Editor region.

(3) Editor toolbar

Frequently used commands of IL are accessible while working in the IL editor.

– Save Editor

- Check Editor
- Insert new line
- Cross references
- Find next cross reference
- Find previous cross reference
- User FB variables (Active only for the configuration of user function blocks)

(4) Editor region

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 Acceptable data types for IL operators and functions on page 165). In the case of function block, this field is assigned automatically following block selection (see Insert function blocks into an IL program on page 178).
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 Insert function blocks into an IL program on page 178).

Parenthesis depth ()

When parenthesizing logical operators, a number 1 ... 8 appears here, which indicates the depth of parenthesis (see Change the number of inputs to function blocks on page 179).

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.

(5) Status bar — The status bar indicates the name of the program which is being edited and name of the user.

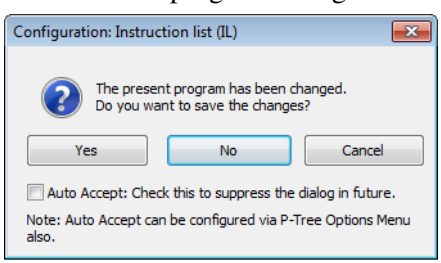
6.2.2 Modify default settings

Auto Accept

Select **Auto Accept** to automatically save any changes in the current editor before switching to another editor.

> Options > Auto Accept

If the option is not enabled, the following dialog box appears for confirmation with each editor or program change:



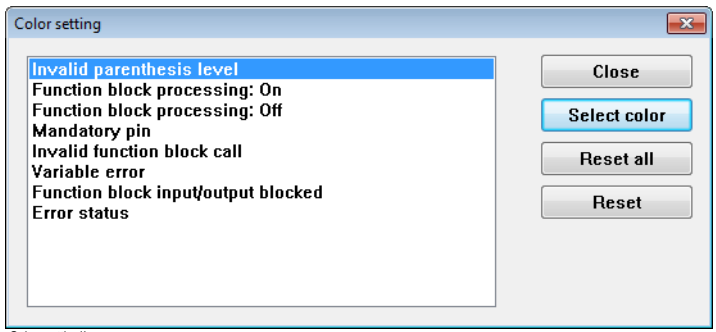
auto_accept_IL_us.png

Adjust colors

> **Options** > **Colors...**

> Select object of which the color is to be changed (e.g. Mandatory pin)

> **Select color** > select required color



Color_settingIL_us.png

Select color The color for the selected object can be chosen. The current color is marked.

Reset The color of the selected object returns to the default value.

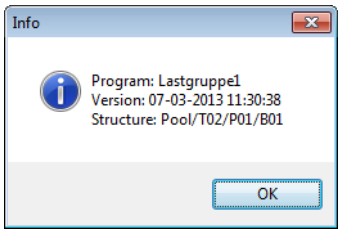
Reset all The colors of all objects are reset to the default values.

6.2.3 Display program information

Program version and position in the project structure

> **Options** > **Version**

The program name, date of last program modification as version identification and the structure path in the project tree are shown.
The structure path can be displayed in a **long** or **short format**, as set in the **Options** menu of the project tree.



di0415us.png

Program state

The **status bar** indicates the name of the program which is currently being edited, the position in the project tree, the current user and license information.

6.2.4 Define favorites list

Functions and function blocks that are required frequently for configuration can be grouped together in a separate list and block menu for easy access.

> **Options > Define favorites list**

For details refer to Specify favorites list on page 110.

6.3 Edit 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.

Content of one editor can be pasted to another editor of same type (For example, IL editor content can only be pasted in into IL editor not into FBD or ST). Also see Section 1, Variables and Section 2, Tags.

6.3.1 Acceptable data types for IL operators and functions

The data types which are possible in Freelance Engineering 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***.

	Output										
Input	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	–	–	–	–	–	–	–	–	–	–	

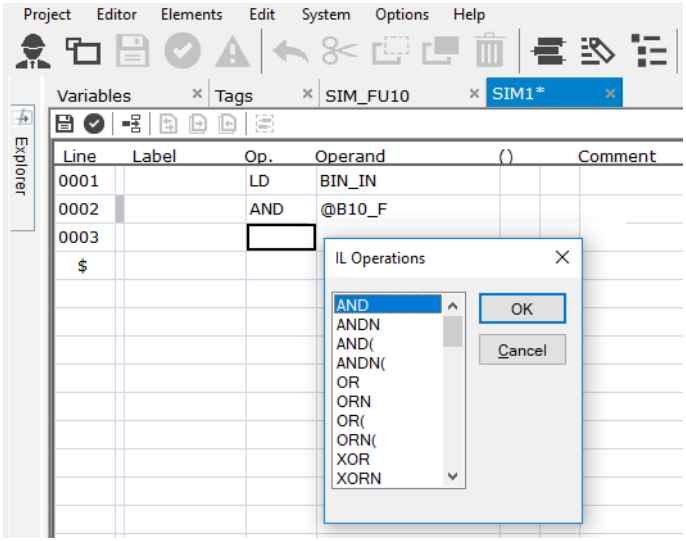
Blocks: TO = *_TO_xx
PA = PACK
EX = EXTRACT

Enter 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#). The possible data types are described in Section 1, Variables.

6.3.2 Call IL operators



tm002us.png

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.

Operators to load and save 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.

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 Overview of IL operators on page 176.

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.

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	)	

IL without parentheses with all intermediate variables		IL without parentheses intermediate variables reduced		IL with parentheses [operation converted]	
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				

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 Insert function blocks into an IL program on page 178). The relation functions can also be called as blocks, but are not distinguished by this from the operators.

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 Acceptable data types for IL operators and functions on page 165) 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 Insert function blocks into an IL program on page 178).

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.

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	**Whi**le **C**ondition	skips the loop if the accumulator is not logical 1.
RPC	**RePe**at 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	**Whi**le **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

Jumps and program calls

Using the jump operators JMP, JMPC, JMPCN the program can be continued at the point named in the argument: that is 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	Init_Saw		already initialized?
0002		EQ	1		
0003		JMPC	L010		Init done: jmp to L010
0004		LD	0		
0005		ST	Saw_Counter		Initialize Counter
0006		LD	1		
0007		ST	Init_Saw		Set Init flag TRUE
0008		ST	Count_UP		Start with counting up
0009					
0010	L010	LD	Count_UP		Count UP or DOWN?
0011		EQ	1		
0012		JMPC	L050		
0013					
0014		LD	Zaehler		Count DOWN
0015		SUB	1		Decrement counter
0016		ST	Zaehler		
0017		LE	-100		
0018		RETCN			Return, if limit not reached
0019		LD	1		Limit reached
0020		ST	Count_UP		Continue with UP
0021		RET			
0022	L050				
0023	L050	LD	Zaehler		Count UP
0024		ADD	1		Increment counter
0025		ST	Zaehler		
0026		GE	100		
0027		RETCN			Return, if limit not reached
0028		LD	0		
0029		ST	Count_UP		Continue with DOWN
0030		RET			
$					

tm003us.png

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.

For calling function blocks the following calls are available:

CAL unconditional call

CALC call only when accumulator = logical 1

CALNC call only when accumulator = logical 0

Overview of IL operators

Operator	Description
AND	Accumulator AND argument to accumulator (= Accu)
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 accu is equal to argument, logical 1 to accu, otherwise logical 0.
NE	If accu is not equal to argument, logical 1 to accu, otherwise logical 0
GT	If accu is greater than argument, logical 1 to accu, otherwise logical 0

Operator	Description
GE	If accu is greater than or equal to argument, logical 1 to accu, otherwise logical 0
LT	If accu is less than argument, logical 1 to accu, otherwise logical 0
LE	If accu is less than or equal to argument, logical 1 to accu, 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

Operator	Description
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

6.3.3 Insert 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 get a name, a comment and a parameter dialog. When they are called, a fixed block of IL lines is inserted into IL ahead of the selected list position. One line is reserved for each input and output. 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 pins) are highlighted in color.

Certain argument fields have a gray background if the relevant input has already been occupied in the parameters dialog 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. In addition, for functions which are available as an operator and as a block, this mark shows that a block is used.

0003		ST	@PLI_1_WDG			
0004		ADD	1			Add via accumulator
0005		ST	@PLI_2_W			
0006	L100					
0007		LD	@PLI_1_WDG			
0008		ADD	2			Add using function block
0009		ST	@PLI_2_W			
0010						
0011		CAL	LIMIT			Analog LIMIT
0012	Name		Limit_tag			tag name
0013	S-Text					
0014	EN					
0015	IN		@IN267			Input using var IN267
0016	ERR		@Err_State			output ERR to var Err_State
0017	OUT		@Limit_Out			output of FB to var Limit_Out
0018	ENO					
0019	PARA-DISP		@@@@@			
$						

tm004us.png

The parameters dialog which belongs to the named block is selected as follows:

Specify parameters for function blocks

> Double-click the field PARA-DISP.

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

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 @@@@@.

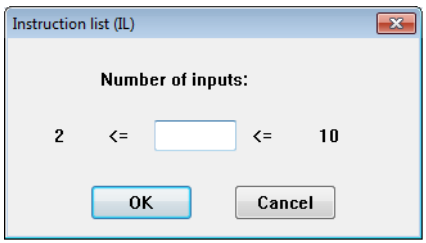
See also Section 5, Function Block Diagram (FBD), Description of FBD program elements on page 124.

Change 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.



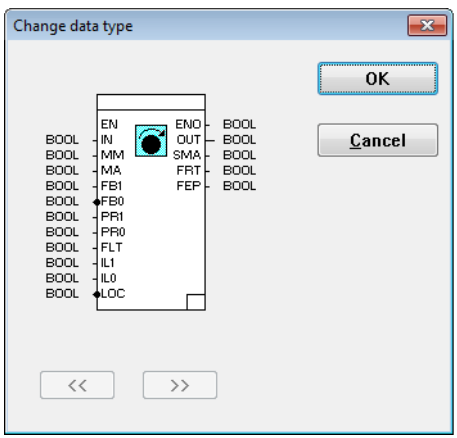
di0413us.png

Change the data type of inputs and outputs

Some functions can handle different data types as input and output values.

> Select block > **Edit** > **Change data type**

> Set and enter the required data type with **>>** and **<<.**



di0131us.png

In the selection dialog the data type of the block terminals are displayed in text and graphically. The data types of the connected variables must match the selection.

For more information, refer to ***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.

6.3.4 Cross references

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

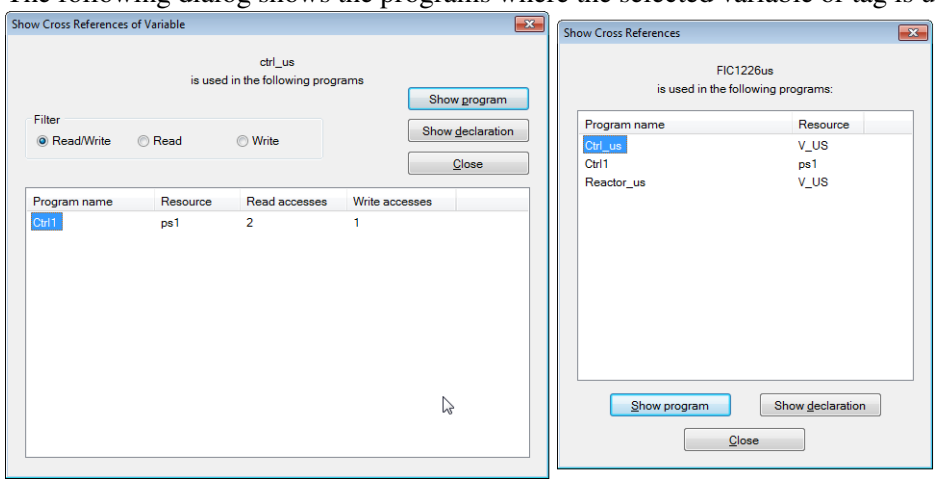
> Select a variable, IO/O component or tag

> Edit > Cross references

or

> F5 key

The following dialog shows the programs where the selected variable or tag is used.



CrossRef_us.png

In contrast to the variables, for the tags no read or write access is defined.

Show program

For a variable:
Call a program with prior selection of these variables, or call the

module with prior selection of the I/O component.
For a tag:
Call the program with prior selection of this tag, or call the module in the hardware structure.

Show Declaration

For a tag, the tag list is called, for a variable the variable list is called. If an I/O component is used directly in the program, the I/O editor of this component is opened.

Filter A filter enables only those variables to be displayed for which read-only access or write-only access exists in the programs concerned.

After activation it is possible to branch to the programs listed as cross references.

Show next / previous cross reference

> Select a variable > **Edit** > **Cross references** > **Find next** or **Find previous**

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

6.3.5 Program administration functions

Save the program

> **Project** > **Save Tab**

The program is saved without exiting. Programs that are not correct can also be saved and then completed at any time.

If the project is not saved in the project tree on closing or before, changes made to the program are ineffective.

Document the program

> Project > Documentation

The editor for documentation is opened as a separate tab in the right pane. This can be closed using the Close button available at the right side of the opened tab, and reopened later from the main menu.

Documentation administration is opened. This is where user-specific project documentation is defined and output. For a description, see ***Engineering Manual System Configuration, Documentation***.

Program header

> Project > Header

A program-specific short comment can be added to the program documentation header, or this can be edited.

For Drawing a header / footer see ***Engineering Manual System Configuration, Documentation***.

Edit program comment

> Project > Comment

A longer program-specific comment can be edited here to describe the functionality. For a description, see ***Engineering Manual System Configuration, Project manager***.

Print

> Options > Print

The contents of the screen are output to the standard printer.

Plausibility check

> **Editor > Check**

All inputs relevant to operation are checked for syntactical and contextual correctness. Errors, warnings and notes that are found are displayed in an error list. If the plausibility check detects errors, the processing state of the program is **implausible**.

The processing state of program elements that are newly entered, copied or moved is **implausible**.

This plausibility check reviews the accuracy and consistency of the program itself. To test the correctness in the project context call plausibility from the project tree. See Engineering Manual System Configuration, Project Tree, plausibility.

Error list

> **Editor > Show error list**

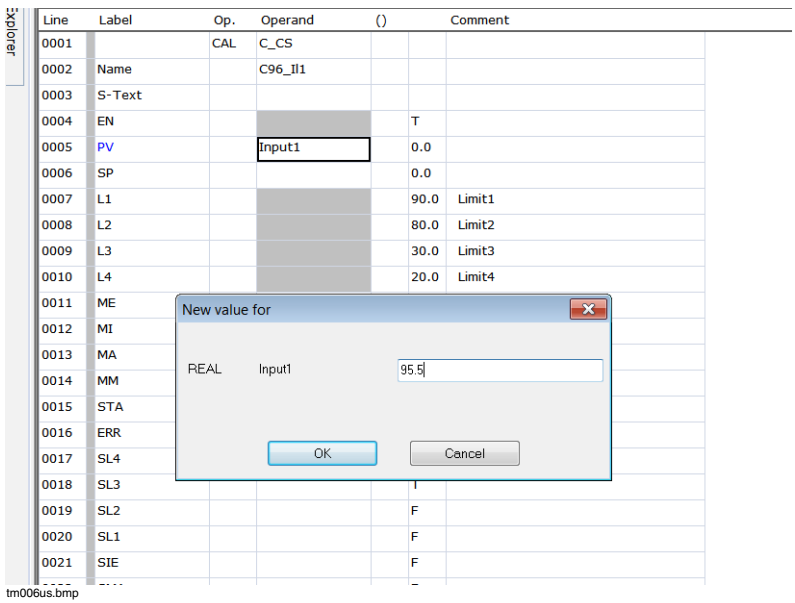
Any errors present in the program is displayed in the error list. Double-click a check message to jump to the line in the program that caused this error.

See also ***Engineering Manual System Configuration, Project tree***.

6.4 Commissioning 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.

If the program editor is opened in commissioning mode, that for showing the live value, CPU load can raise approximately up to by 15%.



Values can be written once within a processing sequence.

> Select variable

> **Window** > **Write value** > Enter value > **OK**

or

> Context menu > **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.

7 Ladder Diagram (LD)

7.1 General Description – Ladder Diagram

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.

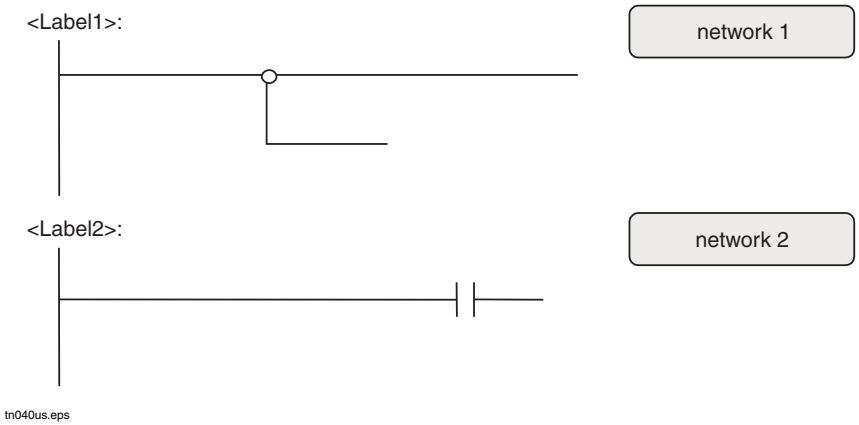
The work area of an LD program is structured over 10x10 pages. An individual page can be reached by vertical and horizontal scrolling. A raster is applied to the entire work area. The breaks between pages are indicated by a dashed line. The program documentation, which is output on a page-for-page basis, 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.



7.1.1 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.

Within the individual networks the signal flow lines are edited with the left mouse button.

As an extension of the IEC language definition, variables and their components of the structured data types may be used.

After loading the programs, in commissioning mode the Editor can be activated if there is an existing connection to the process stations. The current values in the FBD program is displayed. For further details see ***Engineering Manual System Configuration, Commissioning***.

7.1.2 Create an LD program

A LD 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**

> LD program from “Object selection”

> Enter a program name and optionally a short comment

Each new LD program has a blank graphic area, the check state incorrect and the creation date as the version identifier.
The name and the short comment of the program list (PL) are taken over and preset as program name and short comment; both can be changed easily.

Content of one editor can be copied and pasted to another editor of same type (for example, LD editor content can only be pasted in into LD editor not into FBD or ST).

7.1.3 Copy an LD program

> Select program to be copied from project tree > **Edit** > **Copy** or **CTRL+C**

> Select position to which program is to be copied

> **Edit** > **Paste** or **CTRL+V**

> Depending on chosen position, select **Above**, **Below** or **Level**

> Enter new program name

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

7.1.4 Delete an LD program

> Select program to be deleted from project tree > **Edit** > **Delete**

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

7.1.5 Call the LD program editor

A program can be opened by selecting the LD object in the project tree. This can be accessed from the **Edit** menu or by double-clicking the program.The LD Program is opened as a separate tab in the right pane. It can be closed using the Close button available at the right side of the opened tab

> Project tree > **Edit** > **Program**

or

> Double-click the program

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

7.1.6 Close LD program

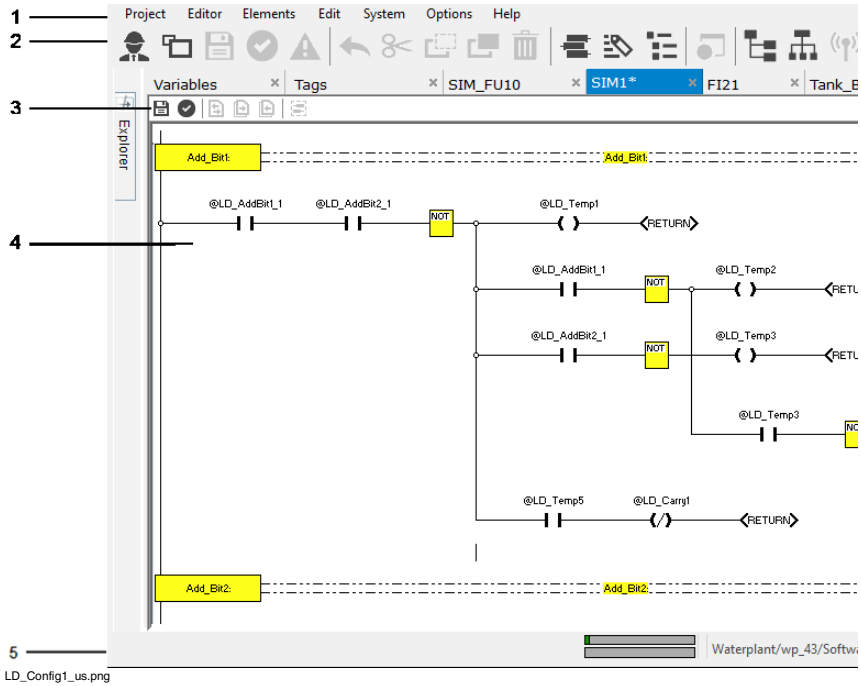
> Editor > Close

Closes the active LD tab.

7.2 Representation of the Ladder Diagram

7.2.1 User interface of the LD editor

The configuration interface of an LD program comprises:



LD_Config1_us.png

(1) Menu bar The menu entries are adapted to the active window or editor in Freelance Engineering.

(2) Common toolbar

The common toolbar is accessible from the Project Explorer and the Editor region.

(3) Editor toolbar

Frequently used commands of LD are accessible while working in the LD editor.

- Save editor
- Check editor
- Cross references

- Find next cross reference
- Find previous cross reference
- User FB variables (active only for the configuration of user function blocks)

(4) Graphic region

The function blocks and signal flow lines are configured in the graphic region of the LD program.
The graphic region 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 elements of the LD program only within this raster. The visibility of 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 not to position objects on the dashed lines, as they would be split up over several pages in the documentation.

(5) Status bar The status bar indicates the name and the page of the program which is being edited and name of the user.

7.2.2 Modify default settings

Auto Router

If the **Auto Router** function is enabled, moving one or more objects automatically adjusts the connection lines. Furthermore, the simplified line drawing mode is activated.

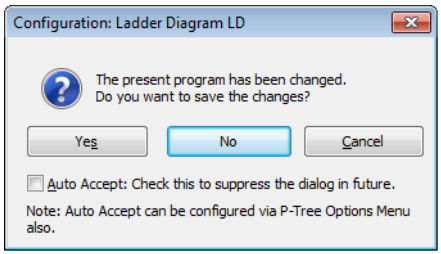
> Options > Auto Router

Auto Accept

Select **Auto Accept** to automatically save any changes in the current editor before switching to another editor.

> Options > Auto Accept

If the option is not enabled, the following dialog box appears for confirmation with each editor or program change:



auto_accept_LD_us.png

Switch the raster on and off

> **Options > Raster on**

All the elements in a LD sheet are positioned within a raster. This positioning raster is made visible by this menu selection, if it was hidden and vice versa. The setting is standard for all LD sheets in the project.

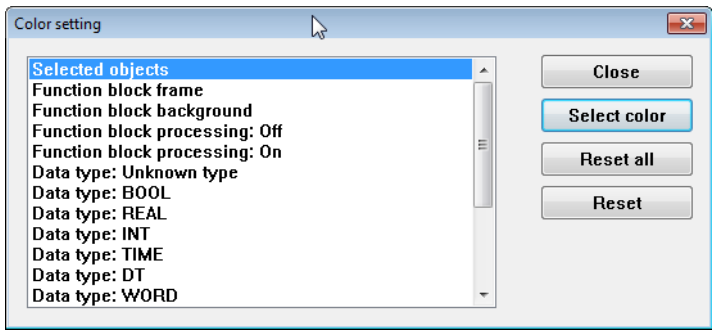
The setting that was saved for the last program to be edited is offered as a default. The spacing of points in the raster cannot be altered.

Adjust colors

> **Options** > **Colors...**

> Select object for which the color is to be changed(e.g. Function block frame)

> **Select color** > select the required color



Color_settingLD_us.png

Select color The color can be chosen for the selected object. The current color is marked.

Reset all All object colors are reset to the defaults.

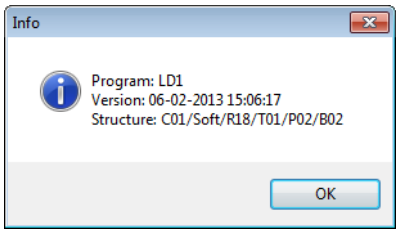
Reset The color of the selected object is reset to the default.

7.2.3 Display program information

Program version and position in the project structure

> **Options** > **Version**

The program name, date of last program modification as version identification and the structure path in the project tree are shown.
The structure path can be displayed in a **long** or **short format**, as set in the **Options** menu of the project tree.



tn032us.png

Program state

The **status bar** indicates the name and current the page of the program which is currently being edited, the position in the project tree, the current user and license information.

Editor position (4,1)
Shows the currently edited page (line, column), here the fourth page horizontally and first page vertically.

7.2.4 Define favorites list

Functions and function blocks that are required frequently for configuration can be grouped together in a separate list and block menu for easy access.

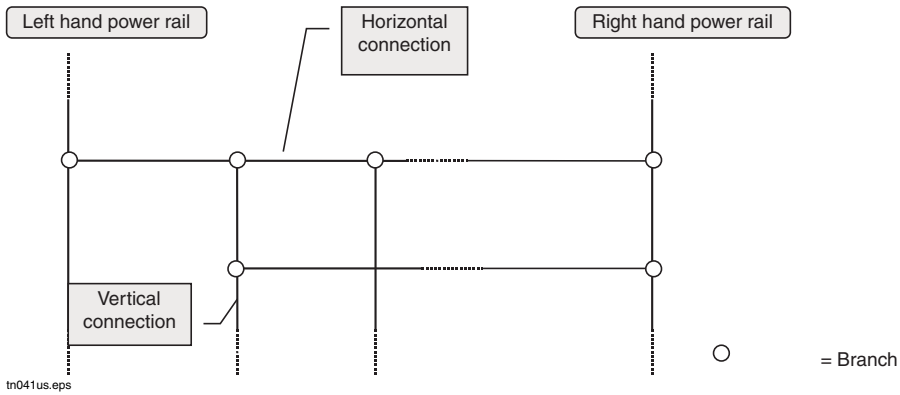
> **Options > Define favorites list**

For details refer to Specify favorites list on page 110.

7.3 Description of the Ladder Diagram elements

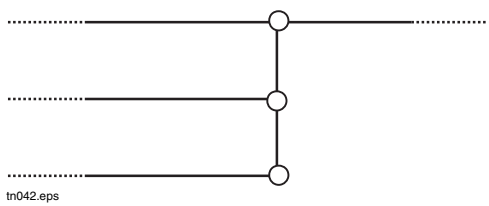
7.3.1Connections and lines

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



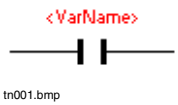
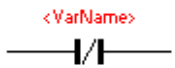
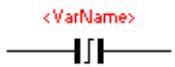
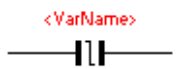
Function	Description
Horizontal connection	Transports the state at the left-hand end to the right-hand end.
Vertical connection	Links all the states on the left-hand horizontal connection with a logical (inclusive) OR and applies the result to the horizontal connections on the right-hand side [1] (wired 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.



7.3.2 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
 tn001.bmp	**Normally-opening contact** The contact is switched when the assigned variable is TRUE
 tn002.bmp	**Normally-closing contact** The contact is switched when the assigned variable is FALSE.
 tn003.bmp	**Positive transition-sensing contact** The contact is switched when the assigned variable has a positive transition.
 tn004.bmp	**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

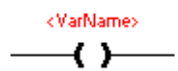
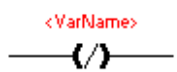
The state of the right-hand side of a **negative transition-sensing contact** can be obtained from the following table:

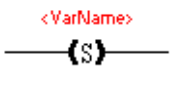
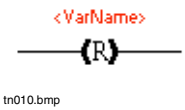
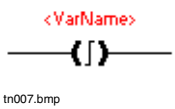
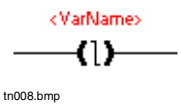
		Previous state of the right-hand side <VarName>	
		0	1
Current state of the right-hand side <VarName>	0	0	State of the left-hand side
	1	0	0

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.

7.3.3 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
 tn005.bmp	**Normal coil** Applies the state of the left-hand connection to the assigned Boolean variable and to the right-hand connection.
 tn006.bmp	**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

Symbol	Description/Function
 tn009.bmp	**Set coil** The assigned Boolean variable is set to TRUE if the state of the left-hand connection is TRUE, otherwise the Boolean variable is left unchanged.
 tn010.bmp	**Reset coil** The assigned Boolean variable is set to FALSE if the state of the left-hand connection is TRUE, otherwise the Boolean variable is left unchanged.
 tn007.bmp	**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 FALSE and the current state is TRUE, then the value TRUE is assigned to the assigned Boolean variable.
 tn008.bmp	**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 TRUE and the current state is FALSE, 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.

7.3.4 Variables and constants

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

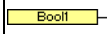
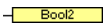
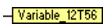
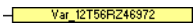
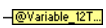
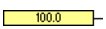
A short and a long rectangle can be selected to display the variable name and/or constants value.

The short version can display 10 characters. If the space in the rectangle is too small to display the complete label length, the overflow is indicated by '….'. The complete label is displayed as a tool-tip. Alternatively the long version can be selected for a permanent display of the complete label length.

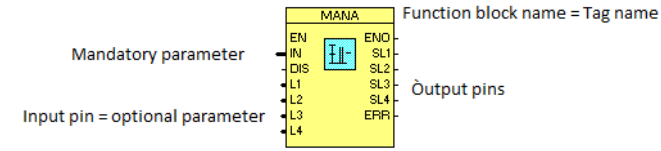
Variables can be read and written either via the process image or directly. Reading or writing via the process image is indicated by @.

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. As long as the variable is not connected with a line, access can be switched between r**ead** and **write** mode via the short-cut menu with **Toggle read access**.

Symbol	**Description/function**
 tn011.bmp	**Variable for reading**
 tn012.bmp	**Variable for writing**
 tn019.bmp	**Short version** At least 10 characters can be displayed Overflow indication '…'
 tn020.bmp	**Long version** Max. possible label length
 tn018.bmp	Read/write via **process image**
 tn023.bmp	**REAL Constant**

7.3.5 Function blocks



tn033us.png

Frame
The block frame limits the selection area of the block. The color indicates whether the block is selected or not. To change the used color, refer to Adjust colors on page 195.

Function block name
Unlike the functions, all function blocks are displayed with a **tag name** (max. 16 characters). All block names are included in the system-wide **tag list**. The font color used for the function block name is used for identifying its processing 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 abbreviation 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 or specified.

Mandatory/ Optional pins
Mandatory pins require the supply by a signal flow line in order to enable the block to operate work correctly, while this does not apply for optional pins. To distinguish the connector pins, the optional pins are shown shorter. Some optional pins disappear, if they are configured with fixed value in the parameter dialog.

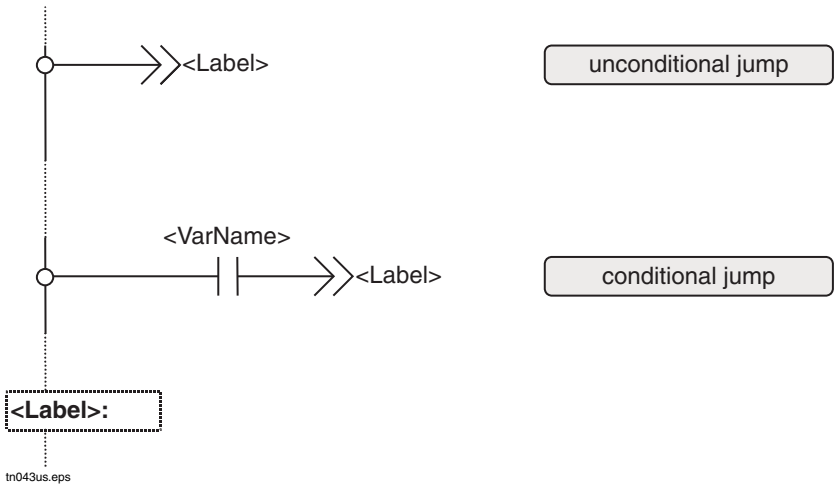
Terminal identifier
In a function block each input/output pin also has a code that represents the function of the pin, e.g. **EN** for **enable**.

7.3.6 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.

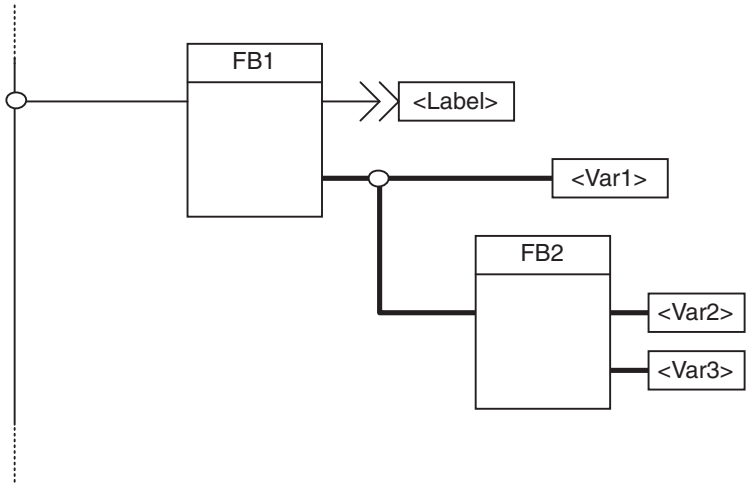
The targets of jumps are designated by a label. Labels are thus local to a particular program.



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

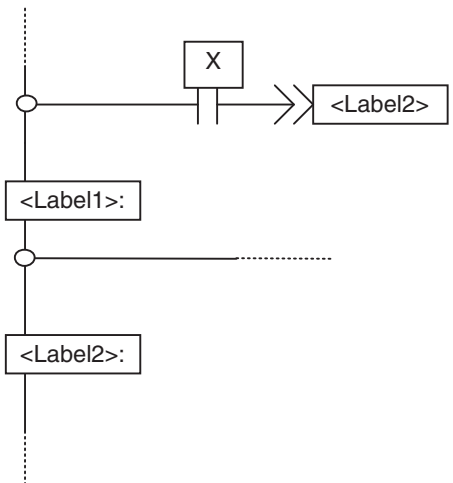
Example

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.



tn044.eps

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



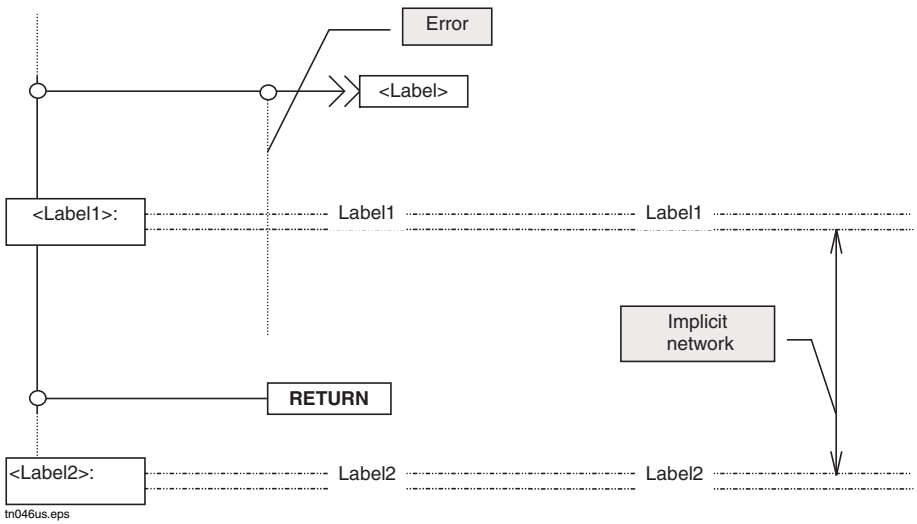
tn045.eps

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

7.3.7 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.



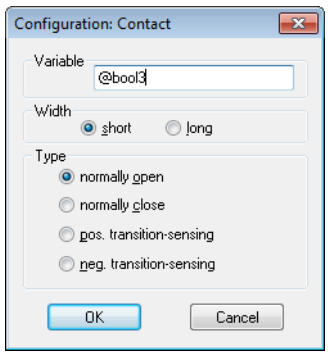
Implicit networks are defined through labels. An implicit network begins at a label or at the beginning of an LD program, and ends at the next label or at the end of the program.

7.4 Parameterize Ladder Diagram elements

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

> **Edit** > **Parameters**
> or
> > Double-click the element
> or
> > Right-click to open context menu > **Parameters**

7.4.1 Parameterize a contact



tn036us.png

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

Width

short/long In the short rectangle about 10 characters can be displayed for the assigned variable. If the space in the rectangle is too small to display the complete label length, the overflow is indicated by '….'. The complete name is displayed as ToolTip. Alternatively the long rectangle can be selected for a permanent display of the complete name.

Type

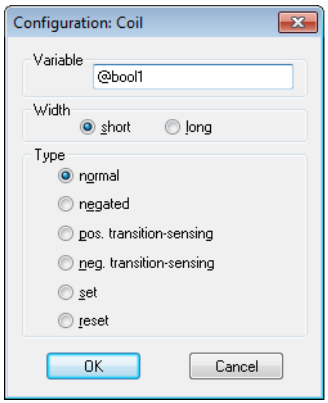
normally open The contact switches when the assigned variable is TRUE.

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.

7.4.2 Parameterize a coil



tn025us.png

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

Width

short/long In the short rectangle about 10 characters can be displayed for the assigned variable. If the space in the rectangle is too small to display the complete label length, the overflow is indicated by '….'. The complete name is displayed as ToolTip. Alternatively the long rectangle can be selected for a permanent display of the complete name.

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.

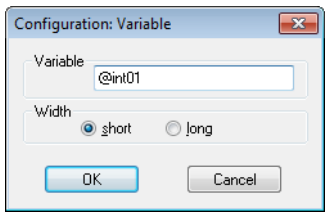
pos. transition-sensing
If there is a positive transition at the coil input, then the assigned variable is set to TRUE. Otherwise it is given the value FALSE.

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.

7.4.3 Parameterize a variable



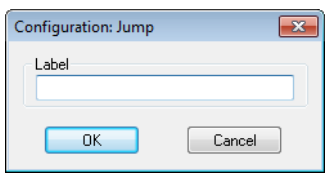
tn026us.png

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

Width

short/long In the short rectangle about 10 characters can be displayed for the variable name. If the space in the rectangle is too small to display the complete label length, the overflow is indicated by '….'. The complete name is displayed as ToolTip. Alternatively the long rectangle can be selected for a permanent display of the complete name.

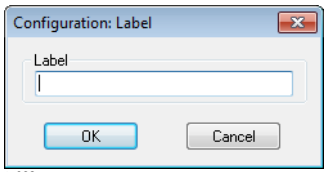
7.4.4 Parameterize a jump



tn027us.png

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

7.4.5 Parameterize a label



tn028us.png

Label The label name.

7.4.6 Parameterize function blocks

When defining parameters for function blocks, the same procedure is used as in the FBD editor. See also Section 5, Function Block Diagram (FBD).

7.5 Edit an LD program

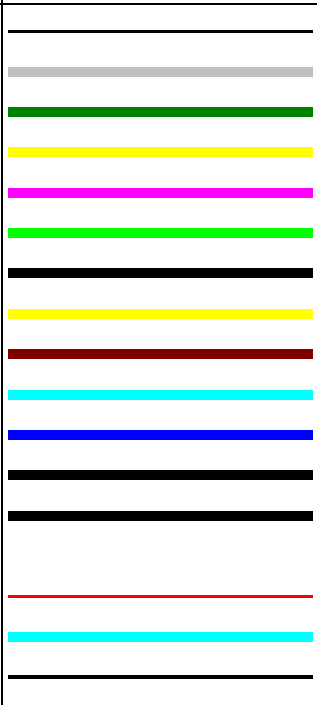
7.5.1 Representation of the signal flow lines

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 Adjust colors on page 195).

The following table shows the connection between data type, edit state, line width and the default color:

Data type/ Processing state	Color	Display	Example
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	



7.5.2 Draw lines

Signal flow lines can either be drawn explicitly or created automatically by the system. To draw the lines explicitly, horizontal and vertical “line sections” are defined; if the signal paths are to be determined automatically it is necessary only to specify the start and end points of the signal flow.

Explicit drawing of signal flow 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

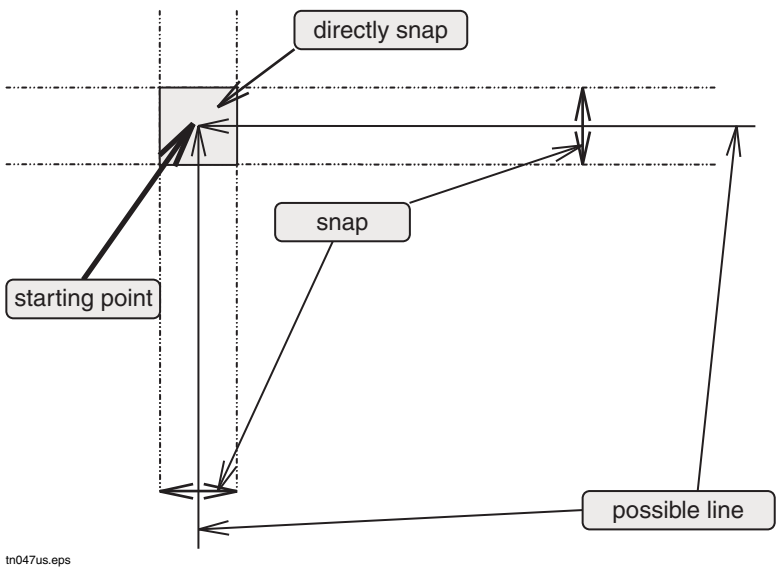
> Right-click (Context menu) > **Draw lines**

The mouse cursor changes into a cross.

A single 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 at the beginning of the line (within the snap) and as long as the line does not cut across any coil, jump, return, block or network boundary.

Each additional click terminates the current line and simultaneously defines 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.

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



Dragging a line

> Right-click with the mouse to define the start and end of a line

or

> Simultaneously press **CTRL** key and left mouse button and draw the line.

A signal flow line can also be drawn directly by pressing the CTRL key and the left mouse button simultaneously. A horizontal or vertical line is defined by releasing the left mouse button.

Releasing the **CTRL** key has the effect of exiting line-drawing mode.

Deactivate draw mode

> Right-click with the mouse

or

> **ESC** key

Automatic drawing of signal flow lines

Start of a signal flow line:
> Click one pin with the left mouse button, drag to the next pin.

End of a signal flow line:
> Release the mouse button.

Auto Router enabled:

This describes the automatic drawing of a pin to pin connection. In order to draw signal flow lines automatically of a pin to pin connection, system will automatically be into the auto connect mode, This will draw the connection lines between the two pins of the LD elements, till the user releases the mouse button. If user clicks on any other part of LD element, element gets selected for other editing options.

Auto Router disabled:

This describes the automatic drawing from everywhere in the LD editor to everywhere else. To draw lines with automatic line drawing, press CTRL +SHIFT and press the left mouse button somewhere in the LD (not on the pin of a block).

The possible path of the signal flow line from the start point to the current cursor position is indicated. When the keys are released the signal flow line is finally defined.

If there is not sufficient free space available in the drawing area, then the signal flow line will not be drawn in.

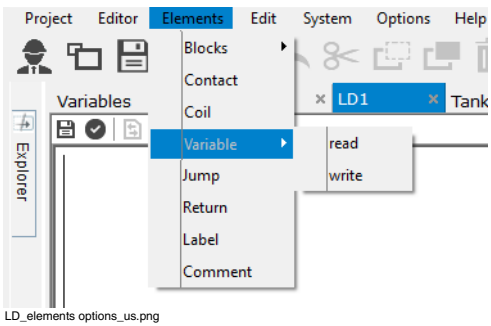
7.5.3 Insert LD elements and function blocks

Variables, blocks and comments can be inserted from the library explorer or from the main menu.

> **Explorer** pane > **Libraries** tab > Select the element to be inserted

or

> **Elements** > Select the element to be inserted



LD_elements options_us.png

After the element to be inserted has been chosen, the cursor takes on the shape of the selected element. The selected element can be positioned into the active tab in the workspace pane 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 should not fit in at the intended destination, the mouse cursor reassumes it normal shape. This clears the selection of the element from the clipboard memory.

If the placing was performed successfully, the outline cursor is retained, and further elements of the selected type can be inserted.

The insertion process is terminated by clicking the right mouse button.

For more information on the Explorer pane, refer to ***Engineering Manual System Configuration***.

7.5.4 Insert or delete columns and rows

In the current program, parts of the configuration can be “pushed apart” by insertion of columns or rows or “pushed together” by deleting columns or rows.

If the cursor is moved at the edge of an implicit network, a small double arrow is superimposed. If the double arrow is in black, it is possible to insert or delete columns or rows at this point. A red arrow means that insertion or deletion is not possible.

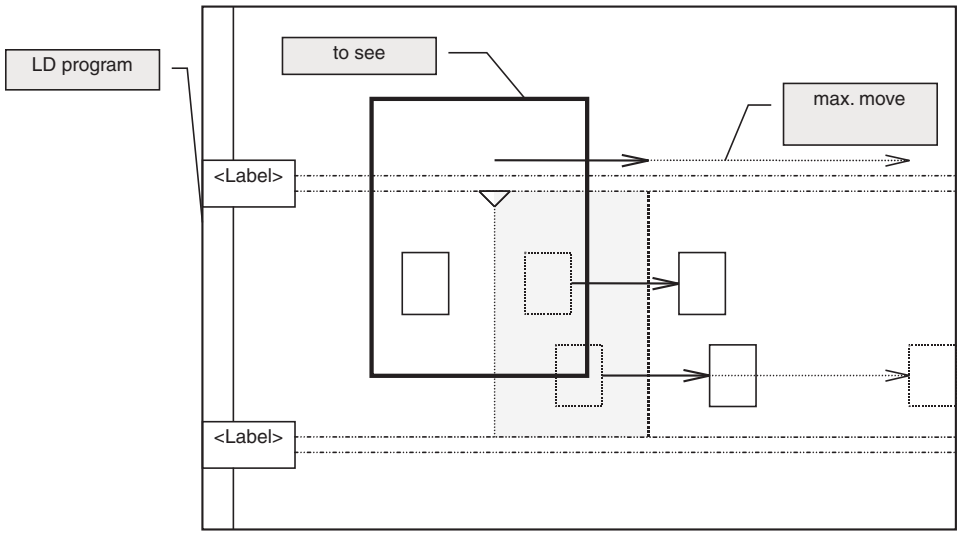
Insert or delete columns

By left-clicking with the mouse on the upper or lower network edge on a black double arrow, a vertical line with two pointed triangles is superimposed at the edge of the hard-clip area. This line can be shifted to the right or left by depressing the left mouse button.

With each shift to the right by one raster unit, a column is inserted into the drawing area and the part of the configuration to the right of the line shifted to the right by one raster unit.

With each shift to the left by one raster unit, a column is deleted from the drawing area and the part of the configuration to the right of the line shifted to the left by one raster unit.

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 with a network element other than a horizontal connection. The diagram below should further clarify the procedure for inserting columns.



When columns are inserted or deleted, horizontal connections are extended or reduced accordingly.

Insert or delete rows

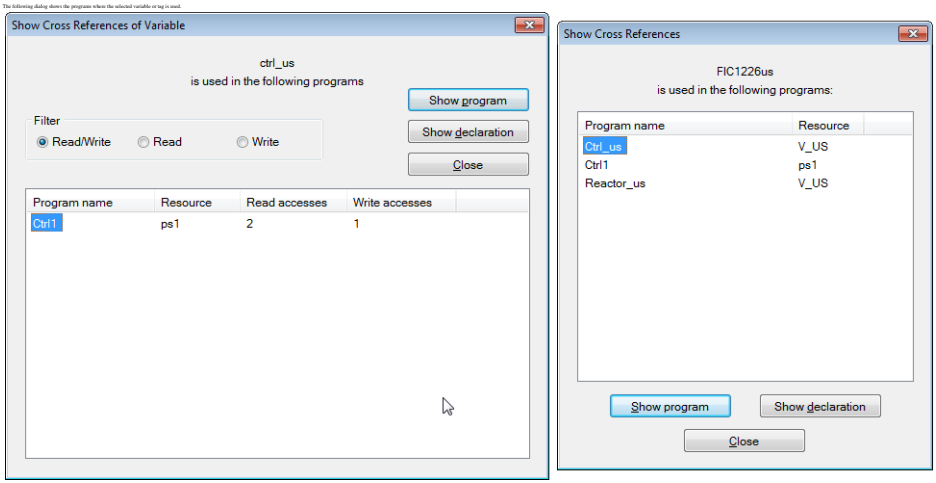
The insertion of rows corresponds to the insertion of columns. The movement markings run in a horizontal orientation. When rows are inserted or deleted, vertical lines are extended or reduced accordingly.

7.5.5 Cross references

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

> Select a variable, I/O component or tag

> **Edit** > **Cross references** or **F5** key



CrossRef_us.png

In contrast to the variables, for the tags no read or write access is defined.

Show program

For a variable:
Call a program with prior selection of these variables, or call the module with prior selection of the I/O component.

For a tag:
Call the program with prior selection of this tag, or call the module in the hardware structure.

Show Declaration

For a tag, the tag list is called, for a variable the variable list is called. If an I/O component is used directly in the program, the I/O editor of this component is opened.

Filter

A filter enables only those variables to be displayed for which read-only access or write-only access exists in the programs concerned.

After activation it is possible to branch to the programs listed as cross references.

Show next / previous cross reference

> Select a variable > **Edit** > **Cross references** > **Find next** or **Find previous**

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

7.5.6 Block operations

Block selection, selection of several program elements

> Drag the mouse to form a rectangle in the graphic area and select all the graphic elements lying wholly within this rectangle.

or

> Press and hold **SHIFT** key > right-click the elements to be selected

All elements that are completely enclosed within the frame are selected simultaneously and displayed accordingly. In the case of signal flow lines, this applies to all sections that lie entirely within the frame. Color for selected elements can be changed via the Options menu (see Adjust colors on page 195).

Copy

> **Edit** > **Copy**

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.

Content of one editor can be pasted to another editor of the same type (For example: LD editor content can only be pasted in into LD editor not into IL or ST). This will enable user to modify the programs easily.

When function blocks are copied, the parameter data remain unchanged. However, the tag name is deleted for the copy, as it must be unique.

Cut / Delete

> **Edit** > **Cut** or **Delete**

After the selected elements have been cut, they can then be re-inserted in the program using **Paste**. Cutting has the effect of overwriting any elements held in the internal store at the time.

If elements are deleted, they can only be pasted directly subsequently using **Undo**, they cannot be pasted at a later time. 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.

Paste

> **Edit** > **Paste**

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

Move block

The following options 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

or

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 movement in a horizontal and vertical direction.

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.

While the selected elements are being moved to a new position their outlines remain visible. Until then blocks that are not assigned mandatory parameters remain in the “incorrect” status. Parameters already configured are retained.

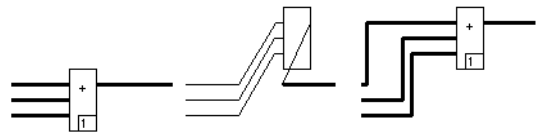
Move block with existing links

If the existing links are to be retained when a block is moved, proceed as follows:

Auto Router enabled:
> Click on a selected block and drag the block to destination.

Auto Router disabled:
Select the object, then press the left mouse button. Press **CTRL** + **SHIFT** and drag the object to destination.

To move a block without existing links:
Deactivate **Auto Router**, click a selected element and drag the element to destination. The block will be moved without existing links.



di0141.bmp

Representation of a function block before, during and after being moved with existing links.

Import block

> **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. Selecting these variables enables their definition in the current project.

Export block

> **Edit > Export block**

The selected elements of a current LD sheet can be exported to a file. An “Export LD box” dialog box appears, containing a list of all previously-exported files in the most recently selected export directory.

Tag names of the blocks selected are not exported.

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.

7.5.7 Program administration functions

Save the program

> **Project** > **Save Tab**

The program is saved without exiting. Programs that are not correct can also be saved and then completed at any time.

If the project is not saved in the project tree on closing or before, changes made to the program are ineffective.

Document the program

> **Project** > **Documentation**

The editor for documentation is opened as separate tab in the right pane. It can be closed using the Close button available at the right side of the opened tab, and reopened later from the main menu.

Documentation administration is opened. This is where user-specific project documentation is defined and output. For a description, see ***Engineering Manual System Configuration, Documentation***.

Program header

> Project > Header

A program-specific short comment can be added to the program documentation header, or can be edited.

For drawing header / footer, see ***Engineering Manual System Configuration, Documentation***.

Edit program comment

> Project > Comment

A longer program-specific comment can be edited here to describe the functionality. For a description, see ***Engineering Manual System Configuration, Project manager***.

Print

> Options > Print

The contents of the screen are output to the standard printer.

Plausibility check

> Editor > Check

All inputs relevant to operation are checked for syntactical and contextual correctness. Errors, warnings and notes that are found are displayed in an error list. If the plausibility check detects errors, the processing state of the program is **implausible**.

The processing state of program elements that are newly entered, copied or moved is **implausible**.

This plausibility check reviews the accuracy and consistency of the program itself. To test the correctness in the project context call plausibility from the project tree. See Engineering Manual System Configuration, Project Tree, plausibility.

Error list

> **Editor > Show error list**

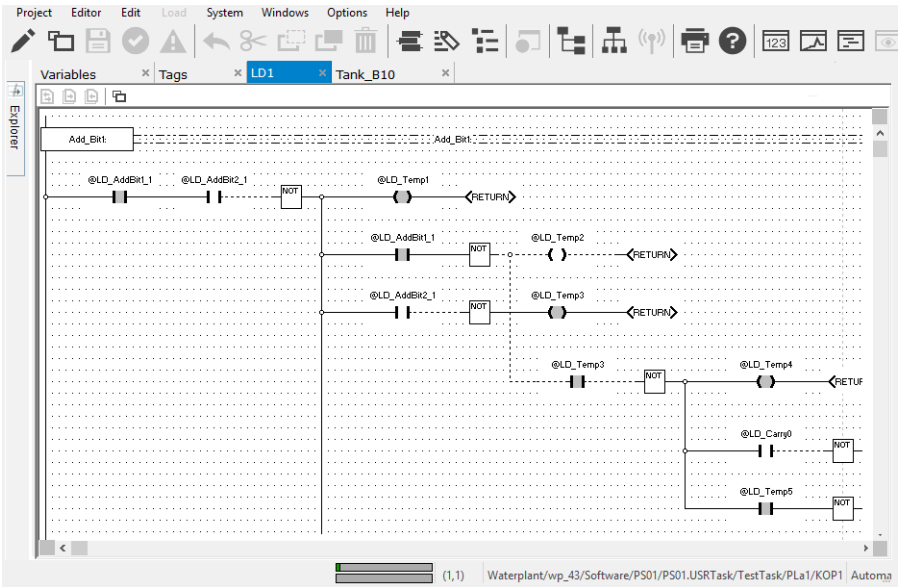
Any errors present in the program is displayed in the error list. Double-click a check message to jump to the line in the program that caused this error.

See also ***Engineering Manual System Configuration, Project tree***.

7.6 Commissioning the Ladder diagram (LD)

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

If the program editor is opened in commissioning mode, that for showing the live value, CPU load can raise approximately up to by 15%.



LD_Commissioning_us.png

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.

> Right-click the 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.

8 Structured Text (ST)

8.1 General Description – Structured Text

Structured text is a text-oriented program language of IEC 61131-3.

Program processing is determined by statements

All functions and function blocks in Freelance Engineering can also be called up in ST programs. The scope of the functions is partly covered by the ST operators. Function blocks can be used after declaration in the ST program. They are parameterized in the same way as in the ladder diagram or in the function block diagram.

Unlike the function block diagram (FBD), the functional scope of the structured text is extended by conditional statements and loop statements that are called up by corresponding keywords.

The processing sequence is obtained by the arrangement of the statements in the ST editor (left to right and top to bottom). The sequence can only be deliberately altered by inserting loop statements.

There is no limit to the working range of an ST program.

After loading the programs and if there is a connection to the process stations, the ST editor can be called up in commissioning mode. The current values in the ST program can be displayed. See also ***Engineering Manual System Configuration, Commissioning***.

8.1.1 Create an ST program

ST programs are created in the.

> Select target position in the project tree

> **Edit** > **Insert above**, **Insert below** or **Insert next level**

> ST program from “Object selection”

> Enter a program name and optionally a short comment.

Each new ST program has a program body: **`PROGRAM`** *Program name* **`END_PROGRAM`**. The check state is incorrect and the creation date is used as version identifier.
The name and the short comment of the program list (PL) are taken over and preset as program name and short comment; both can be changed easily.

Content of one editor can be copied and pasted to another editor of the same type (for example, ST editor content can only be pasted into ST editor, not into FBD or IL).

8.1.2 Copy an ST program

> Select program to be copied from project tree > **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**

> Enter new program name

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

8.1.3 Delete an ST program

> Select program to be deleted from project tree > **Edit** > **Delete**

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

8.1.4 Call the ST program editor

A program can be opened by selecting the ST object in the project tree. This can be opened from the **Edit** menu or by double-clicking the program. The ST program is opened as a separate tab in the right pane. It can be closed using the Close button available at the right side of the opened tab.

> Project tree > **Edit** > **Program**

or

> Double-click the program

The program is displayed with its current content and can be modified.

8.1.5 Close ST program

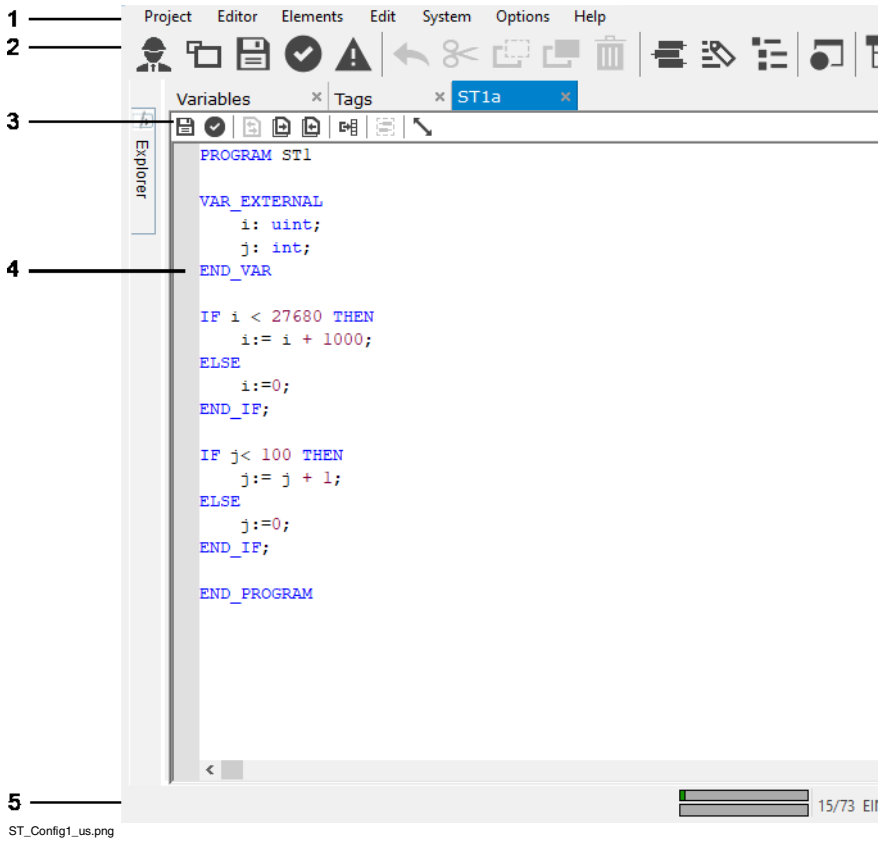
> Editor > Close

Closes the active ST tab.

8.2 Representation of the Structured Text

8.2.1 User interface of the ST editor

The configuration interface of an ST program consists of:



ST_Config1_us.png

(1) Menu bar The menu entries are adapted to the active window or editor in Freelance Engineering.

(2) Common toolbar

The common toolbar is accessible from the Project Explorer and the Editor region.

(3) Editor toolbar

Frequently used commands of IL are accessible while working in the ST editor.

- Save editor
- Check editor
- Cross references
- Find next cross reference

- Find previous cross reference
- Instantiate
- User FB variables
- Toggle breakpoint

(4) Text area — The application is programmed in the ST editor text area. The cursor can be positioned anywhere in the text area. The tabulator width is adjustable. The text area for an ST program is unlimited. There is a mark column on the left of the text field.

(5) Status bar — The status bar indicates the name and the page of the program which is being edited and name of the user.

8.2.2 Syntax coloring

All input in the ST editor is text-oriented. Certain important words in the text are color-highlighted for emphasis. These are:

- Comments
- Keywords
- Keywords not supported (acc. to IEC 61131-3)[1]
- Numeric constants
- Numeric constants are defined in the program flow.
- Symbolic constants
- Symbolic constants are defined in a **`CONST END_CONST`** block.
- Strings
- Blocks that cannot be edited

All other text is shown in black.

1. These include data types such as SINT, for example, that are not supported.

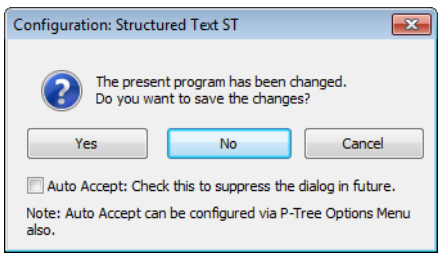
8.2.3 Modify default settings

Auto Accept

Select **Auto Accept** to automatically save any changes in the current editor before switching to another editor.

> **Options** > **Auto Accept**

If the option is not enabled, the following dialog box appears for confirmation with each editor or program change:



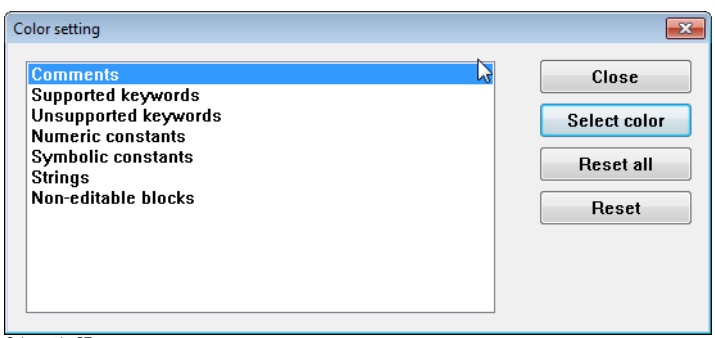
config_ST_us.png

Adjust colors

> **Options** > **Colors...**

> Select the object for which the color is to be changed (e.g. symbolic constants)

> **Select color** > select required color



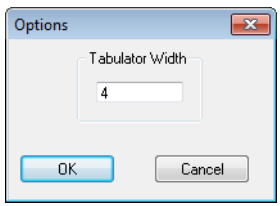
Color_settingST_us.png

Select color	The color for the selected object can be chosen. The current color is marked.
Reset all	The colors for all objects are reset to the default colors.
Reset	The color for the selected object is reset to the default color.

Tabulator width

> **Options > Tabulator width...**

> Enter the desired tabulator width



to003us.png

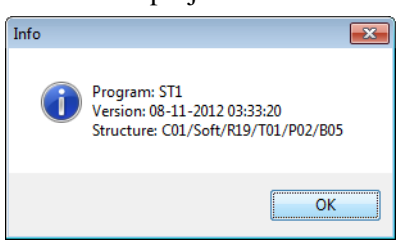
The program code already created is not changed by the alteration in tabulator width. The new tabulator width is used for all subsequent editing steps.

8.2.4 Display program information

Program version and position in the project structure

> **Options > Version**

The program name, date of last program modification as version identification and the structure path in the project tree are shown.
The structure path can be displayed in a **long** or **short format**, as set in the **Options** menu of the project tree.



to004us.png

Program state

The **status bar** indicates the name and the current cursor position of the currently edited program, the edit mode (Insert or Overwrite), the position in the project tree, the current user and license information.

Editing position
(4,1) - Shows the current position of the cursor (row and column).

Editing mode (INS) - Insert
(OVR) - Overwrite

8.2.5 Define favorites list

Functions and function blocks that are required frequently for configuration can be grouped together in a separate list and block menu easy access

> **Options** > **Define favorites list**

For details refer to Specify favorites list on page 110.

8.3 Description of the ST program elements

8.3.1 Language elements

Special symbols and reserved words

Special symbols and words control the flow in an ST program and these must not be used as identifiers in the program. Each special symbol has a particular meaning. ST interprets the following characters as special symbols:

+ - * / & = < > [ ] . , ( ) : ; ’ @ # $

In addition, combinations of special symbols are used as operators and/or delimiters:

:= => Assignment operators

<> <= >= Relational operators

**	Exponentiation operator
..	Subareas
(* *)	Beginning and end of comment

There is no case distinction for reserved words. ST reserves the following words, which are all printed in **boldface** in this manual:

AND
ARRAY
BY
CASE
CONST(1)
DO
ELSE
ELSIF
END_CASE
END_CONST(1)
END_IF
END_FOR
END_FUNCTION_BLOCK(2)
END_PROGRAM
END_REPEAT
END_VAR
END_WHILE
EXIT
FALSE
FOR
FUNCTION_BLOCK(2)
IF
MOD
NOT
OF
OR
PROGRAM
REPEAT
RETURN
THEN
TO
TRUE
UNTIL
VAR
VAR_EXTERNAL
VAR_INPUT(2)
VAR_OUTPUT(2)
WHILE
XOR

(1) These keywords are extensions of IEC 61131-3.
(2) These keywords are only used in user-defined function blocks.

All standard data types are also treated as keywords. Names of function block inputs and outputs that are identical with keywords start with an underline character to distinguish them from the keywords, e.g.:

```
pin_dt(_DT => dt1);
```

Identifier

All types, variables, constants, functions, function blocks and arrays are identified by identifiers. Identifiers within an ST program are words with

- more than one letter (leading numeric characters are allowed)
- one letter, except 'E' or 'e'
- the letter 'E' or 'e' at the beginning or end.

Examples of identifiers:

`123block`	Valid identifier
`123e`	Valid identifier
`123e2`	Syntax error
`123.0e2`	Real number
`block-e`	Syntax error

Case is significant in identifiers. Identifiers must not include any special symbols (see Special symbols and reserved words on page 234). Therefore a dash (which could also be a minus sign) must not occur in a identifier. There is no limit to the length of identifiers within ST programs.

The following identifiers are already defined for standard data types:

BOOL	INT	STR64	UINT
BYTE	REAL	STR128	WORD
DINT	STR8	STR256	
DT	STR16	TIME	
DWORD	STR32	UDINT	

There is no distinction between upper and lower case in standard data types.

Identifiers of global variables must also obey the rules for variable names. See Structure of the variable list on page 23. Identifiers for function blocks also follow the rules for tag names. See Call the tag list on page 59. Unlike other programming languages such as ladder diagram and function block diagram, tag names consisting only of numeric characters are not allowed in ST programs.

Constant

A constant declaration declares a identifier which stands for a particular value within the ST program. Constants are declared in a block **`CONST END_CONST`**. The constant is only valid within the ST program.

Constant expressions must not contain any variables or function calls but they can include constants that have already been defined. Example:

```
CONST
  max := 100;
  max2 := max * 2;
END_CONST
```

The value of a constant expression is calculated by code generation. Constants can be used in expression and for defining areas in arrays. Example:

```
VAR
  MyArray: ARRAY [0..max-1] OF int;
END_VAR
```

The data type of constants is only defined when they are used. The defaults for data types must be observed when entering constants.

See Overview of simple data types on page 22.

Integer constants are always processed internally as DINT values. If a different data type is needed in the program flow then an explicit type change must be inserted. Example:

```
CONST
  BITFIELD := 16#c2420000;
END_CONST

VAR
  var1 : UINT;
  var2 : REAL;
END_VAR

var1 := BITFIELD;
(* This statement leads to a plausibility error. *)
(* The constant BITFIELD is greater than MAX-DINT. *)

var1 := to_di(BITFIELD);
(* With the external type change the value of *)
(* BITFIELD is limited to MAX-DINT. *)
```

```
var2 := to_re(to_dw(BITFIELD));
(* Bit field is converted to a REAL value *)
```

Program

An ST program is enclosed between the keywords **PROGRAM** and **END_PROGRAM**.

```
PROGRAM STprg1
;
END_PROGRAM
```

No executable statements may be written before **PROGRAM**. All statements after **END_PROGRAM** will be disregarded. Every ST program needs a program name. The name of the ST program is preset as a program name in the project tree. The program name is a identifier in the ST program.

In the ST program the declarations of constants and variables must be entered at the beginning. The program code follows.

An empty ST program must contain at least one empty statement.

User-defined function block

User-defined function blocks in structured text are included in the keywords **FUNCTION_BLOCK** and **END_FUNCTION_BLOCK**.

```
FUNCTION_BLOCK UFBprg1
;
END_FUNCTION_BLOCK
```

No executable statements may be written before **FUNCTION_BLOCK**. All statements after **END_FUNCTION_BLOCK** will be disregarded. The name of the user-defined function block is a identifier in the ST program. The interface definition is added into the user-defined function block as a non-editable block. In other respects the programming rules for ST programs apply.

Comment

The purpose of comments is to clarify the program code. They are not taken into account when the code is generated. Comments are enclosed in brackets with a star:

```
(* This is a comment *)
```

Comments can be included at any point in an ST program. Nested comments are allowed:

```
(* Comment (*This is a nested comment *) *)
```

Nested comments are an extension of IEC 61131-3.

Program line

There is no limit to the length of a program line. A program line may include more than one statement.

8.3.2 Types

Every declaration of a variable must give the type of the variable. The type defines the value range of the variable and determines the operations that can be executed with it. See also Overview of simple data types on page 22.

Simple types

Simple types define ordered quantities of similar values.

Integer types

Integer values are a partial quantity of the whole numbers. The following integer types can be used in ST programs: INT, UINT, DINT and UDINT.

Two integer values can only be linked via a binary operator (i.e. addition, multiplication etc.) if the types are the same. If the types are different, an explicit type conversion must be used, e.g.:

```
VAR
  myInt: INT;
  myDint2, myDint1: DINT;
END_VAR
  myDint2 := TO_DI(myInt) + myDint1;
```

Bitfield types

Bitfield types define bit fields of differing length. Bitfield types BYTE, WORD and DWORD can be used in ST programs.

Operations on two bit fields via a binary operator (i.e. bit-by-bit AND, bit-by-bit OR etc.) are only possible if the types are the same.

Boolean types

Boolean types can only take on the predefined values **`FALSE`** or **`TRUE`**, Where **`FALSE`** = 0 and **`TRUE`** = 1.

REAL type

The real type is a subset of the real numbers. A real value `n` has three components. Here `m*2e = n`, where m and e are whole numbers.

Strings

A string is a fixed-length sequence of characters. The following string types can be used in ST programs: STR8, STR16, STR32, STR64, STR128 and STR256. Strings are stored in ASCII code.

Structured types

A structured type, identified by the type of structure, contains more than one value. All defined structured data types can be used in ST programs. For a definition of structured data types, see Structured data types on page 54.

The components of structured types can be accessed with *variable name.component name*.

Arrays

Arrays have a set number of components of a single type. Simple and structured types are both permissible data types. The validity range of the data type for the array index is defined as DINT (-2 147 483 648 .. +2 147 483 647). The number of elements of each dimension is determined by the data type of the array index. The start and end of the definition of the range must be within the range of validity of the array index.

Arrays are defined by the keyword **`ARRAY`**, the range and the data type, e.g.

```
ARRAY [0..100] OF REAL;
```

If the component type of a array is also a array, it is treated as a multidimensional array. Up to four dimensions can be used for each array declaration, e.g.

```
ARRAY [0..100, 0..10, -2..2] OF INT;
```

The components of arrays can be accessed with
variable name[Index1, Index2].

The components of structured data type arrays can be accessed with
variable name[Index1, Index2].component name.

Arrays can only be used within ST programs. It is not possible to exchange arrays between different ST programs.

The array index is checked for validity at the time of running in the process station. If the array index is outside the defined range, the task state changes to **not executable**.

8.3.3 Variables and function blocks

Declaration of variables

Variables must be declared before use. The variable declaration states the type of variable. All standard data types and structured data types that have already been defined can be used. The variable name and data type are separated by a colon, e.g.

```
VAR
  x, y: INT;
END_VAR
```

ST programs distinguish between local variables and global variables. If a local variable has the same name as a global variable, the local variable takes precedence.

Global variables

Global variables are also valid outside the ST program. All variables defined in the variable list can be used as global variables in ST programs. New variables can also be created in an ST program. After declaration in the ST program, new variables also have to be entered in the variable list, i.e. they must be instantiated. Instantiate

on page 267 gives further details. Global variables are declared within the keywords **VAR_EXTERNAL END_VAR**, e.g.

```
VAR_EXTERNAL
  TIC1379_PV: REAL;
  TIC1379_MODE: INT;
  TCP_Data_IN01: structTCP12;
  (* structTCP12 is a structured data type *)
END_VAR
```

It is not possible to assign the data type to a list of global variables. Each global variable must be declared individually.

I/O components of hardware objects cannot be declared directly. The hardware object has to be declared as a function block in the ST program.

Local variables

Local variables are only valid within the ST program. The names of local variables only need to be unique within the ST program. The same data type can be assigned directly to a comma-delimited list of variables. Arrays can only be defined as local variables. Initialization is possible at the variable declaration point. Local variables are declared within the keywords **VAR END_VAR**, e.g.

```
VAR
  x, y, z: INT
  F1: ARRAY [0..40] OF REAL;
  TIC1379_OUT: REAL := 10.0;
  a, b, c: INT := 1;
  (* a, b and c are initialized at 1 *)
END_VAR
```

When initializing multidimensional arrays the elements of each individual dimension are given in square brackets. A bracketed pair at the same level is separated by a comma. The initialization

```
VAR
  F2: ARRAY [-1..1, 10..11] OF INT :=[[1,2],[3,4],[5,6]];
END_VAR
```

is equivalent to assignment with the following values:

```
F2[-1, 10] := 1;
F2[-1, 11] := 2;
F2[ 0, 10] := 3;
F2[ 0, 11] := 4;
F2[ 1, 10] := 5;
F2[ 1, 11] := 6;
```

Initial values are assigned to the variables when the program is loaded.

System variables

By definition, system variables are known in the ST program. There is no need to declare them explicitly. The ST program uses system variables as variables with a structured data type, e.g.

```
Date_Time := ps12.DateTime;
```

Inputs and outputs

Inputs and outputs can be defined for user defined function blocks. They are defined in the interface editor of the user defined function block and the definition is displayed within the keywords **VAR_INPUT END_VAR** and **VAR_OUTPUT END_VAR**, e.g.

```
VAR_INPUT    (* declaration of inputs *)
  IN: REAL;
  MD: BOOL;
END_VAR

VAR_INPUT    (* declaration of outputs *)
  IN: REAL;
  STA: INT;
END_VAR
```

Inputs and outputs can only be edited in the interface editor of the user-defined function block.

Keywords must not be used for the names of inputs and outputs.

The names of inputs and outputs should not exceed 3 characters. Longer names are permissible. The long name (>3 characters) must be used in the class definition. If

instances of this function block are used in ST programs the long name is truncated to the first 3 characters.

Function blocks

Like variables, function blocks must be declared before they are used in an ST program. The names of function blocks must be unique throughout the project, i.e. they may only be called up once in the project. In ST programs all standard function blocks and plausible user-defined function blocks can be used (see also Limits of the system on page 257). Local variables and function blocks can be declared in the same block.

All function blocks defined in the tag list can be used in ST programs. New function blocks can also be created directly in an ST program. After declaration in the ST program, new function blocks also have to be entered in the tag list, i.e. they must be instantiated. Instantiate on page 267 gives further details. Function blocks are declared within the keywords **VAR END_VAR**, e.g.

```
VAR
  TI1379: AI_TR;
  TIC1379: C_CU;
  TY1379: AO_TR;
  TI1379_LIN: LIN2;
  (* LIN2 is a user-defined function block type *)
  m: INT;
END_VAR
```

It is not possible to assign the function block type to a list of function blocks. Every function block must be declared individually.

To access the I/O components of hardware objects, the hardware object must be declared as a function block in the ST program in the section **VAR_EXTERNAL END_VAR**, e.g.

```
VAR_EXTERNAL
  DAI02_1_0_1: DAI02;
END_VAR
```

The components are then accessed, with

```
h := DAI02_1_0_1.Ch0;
```

8.3.4 Expressions

Syntax of expressions

An expression is a construct which supplies a value when calculated. Expressions consist of operators and operands. An operand can be a constant, a variable or another expression. Most operators in ST link two operands and are therefore referred to as binary. The remaining operators work with one operand and are therefore referred to as unary.

Binary operators use the common algebraic form, as in `A + B`. A unary operator always immediately precedes its operand, as in `-B`.

Operators

Operators have an order of precedence which governs the sequence of calculation.

Operation	Symbol	Order of precedence
brackets	(expression)	highest
function evaluation	identifier (argument list) e.g. MAX(x, y), LN(a), etc.	
exponentiation	**	
negation complement	- NOT	
multiplication division modulo	* / MOD	
addition subtraction	+ -	
comparison	<, > <=, >=	
equality inequality	= <>	

Operation	Symbol	Order of precedence
Boolean AND Boolean AND	& AND	
Boolean exclusive or	XOR	
Boolean OR	OR	lowest

Calculation of an expression consists in applying the operators to the operands in the sequence defined by the order of precedence. The following rules apply to the calculation of long expressions:

- An operand between two operators of different priorities is always associated with the higher-ranking operator.
- An operand between two operators of the same priority is always associated with the operator to the left of it.
- Expressions in brackets are considered as a single operand and always evaluated first.

The expressions below give the following results:

```
a := 1; b := 2; c := 3; d := 4;
a * b + c              (* Result = 5 *)
a + b * c              (* Result = 7 *)
(a + b) * c            (* Result = 9 *)
a * b + c * d          (* Result = 14 *)
a + b * c + d          (* Result = 11 *)
(a + b) * (c + d)      (* Result = 21 *)
```

Function calls

Functions must be called as elements of expressions that consist of the function name followed by a list of arguments in brackets. All standard functions can be used.

It is not possible to define user-specific functions.

Examples of function calls:

```
SQRT(a)
SIN(a)
MAX(a, b, c)
```

The function call

```
ADD(a, b)
```

has the same value as the expression

```
a + b
```

If constants are transferred exclusively to a polymorphic function as the argument, at least one explicit type change is needed, e.g.

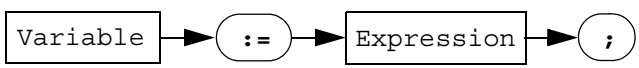
```
CONST
  CstVal := 12;
END_CONST

VAR
  i: INT;
END_VAR

i := add(to_in(CstVal), 42);
```

8.3.5 Statements

Statements are all constructs that declare an action that can be carried out by the process station. Statements must end with a semicolon.

Simple statements

Simple statements are assignments.



Assignments replace the value of a variable with a new value that is given by an expression. This expression can include identifiers of functions that are thereby activated and supply the corresponding values.

An assignment must consist of a variable on the left, followed by the assignment operator :=, followed by the expression to be evaluated. The statement

```
A := B;
```

replaces the value of the variable A by the value of the variable B. Both variables A and B must be of the same data type. The following are valid assignments:

```
x := y + z;
Done := (i >= 1) AND (i<100);
m := 3.0 + SIN(n);
a := feld[i, j].content;
```

The assignment of arrays is not possible. The following assignment leads to a check error:

```
VAR
  Arr1 [1..10] OF BOOL;
  Arr2 [1..10] OF BOOL;
END_VAR

Arr2 := Arr1
```

Global variables can be accessed directly in statements, or via the process image. As in other program editors, an @ must be placed in front of variable names to access the process image:

```
A := SQRT(B);       (* direct access *)
@A := SQRT(@B);     (* access via process image *)
```

Constants in expressions are always calculated internally with the data type DINT. If constants with data type UDINT are appropriate, an explicit type change must be provided, e.g.

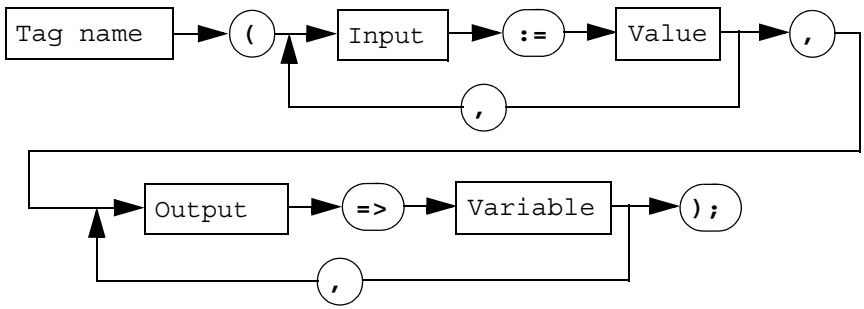
```
VAR
  L_Int: UDINT;
END_VAR
  L_Int := 10;
  L_Int := L_Int * TO_UD(256345984);
```

Unlike other program editors, ST programs no longer require explicit assignment of data types to functions with more than one data type set. The ST editor recognizes the data types needed and automatically assigns the appropriate data type set.

An empty statement consists of just a semicolon.

Function block calls

Function blocks are called by an statement consisting of the name of the function block followed by a list of value assignments in brackets.



The sequence of input and output assignments is not significant. When the function block is called, assignments only have to be supplied to and retrieved from the mandatory pins. Inputs are supplied by := and outputs are retrieved by =>, e.g.

```
VAR
  LI347_LIN: LIN;
END_VAR
VAR_EXTERNAL
  LI347_CH0: REAL;
  LI347: REAL;

END_VAR
  LI347_LIN (IN:= LI347_CH0, OUT=> LI347);
```

Each function block may only be used in one assignment in the ST program.

Optional function block pins can also be accessed outside the function block assignment with
function block name.pin name.

```
VAR
  FIC251: C_CU;
  RATIO, BIAS: REAL;
END_VAR
VAR_EXTERNAL
  FIC251_PV: REAL;
```

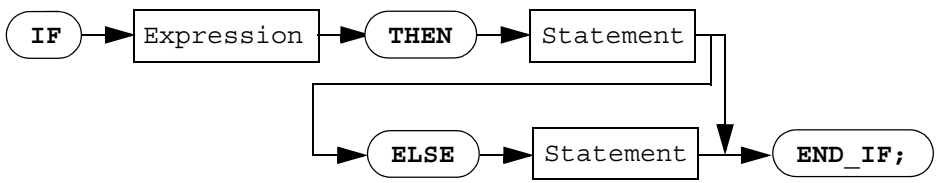
```
  FIC251_OUT: REAL;
  TIC251_OUT: REAL;
  TRD3_ASP: REAL;
END_VAR
  (* supplying inputs *)
  FIC251.SP := TIC251_OUT * RATIO + BIAS;
  (* calling the function block *)
  FIC251(PV:= FIC251_PV, OUT=> FIC251_OUT);
  (* sending outputs *)
  TRD3_ASP := FIC251.ASP;
```

Conditional statements

A conditional statement chooses one of its assignments (or a group of them), of which it is made up, on the basis of a specified condition. Conditional statements are **IF** and **CASE**.

IF statement

The **IF** statement can be shown with the following syntax diagram:



The result of the expression must be of the data type BOOL. If the result is **TRUE**, the part following **THEN** is carried out, if it is not, the part following **ELSE** is carried out. **ELSE** is optional. If this branch is not available, the IF statement has no effect, i.e. it carries nothing out whatever. A couple of examples:

```
IF y<>0.0 THEN
  z := x / y;
ELSE
  z := 3.4e38;
END_IF;
```

```
IF p <> 0 THEN
  a := SIN(b + c);
END_IF;
```

If the statement in an **ELSE** part is another **IF** statement, this can be summarized as **ELSIF**. The following **IF** statement:

```
IF e1 THEN
  s1:=1;
ELSE
  IF e2 THEN
  s1:=2;

  END_IF;
END_IF;
```

is the same as:

```
IF e1 THEN
  s1:=1;
ELSIF e2 THEN
  s1:=2;
END_IF;
```

Serial nesting levels with **IF .. THEN .. ELSE** are syntactically ambiguous. In the following example, it is not possible to determine without doubt which **IF** the last **ELSE** refers to:

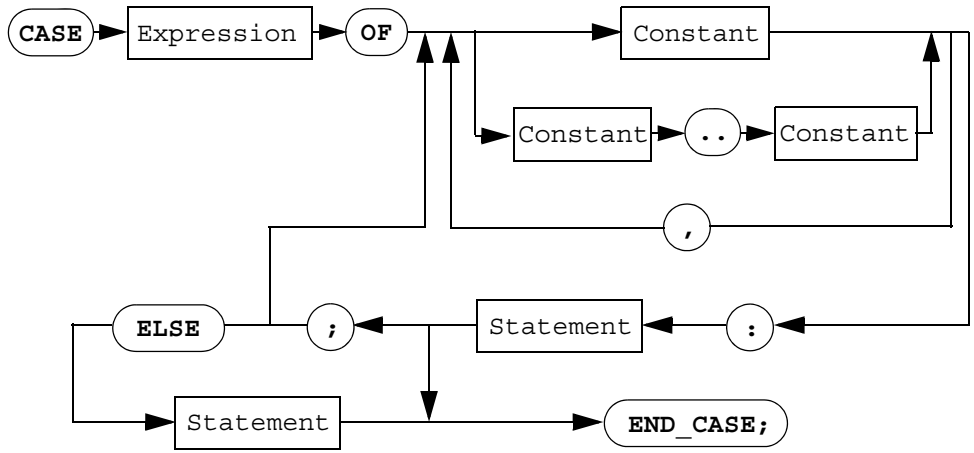
```
IF e1 THEN IF e2 THEN s1:=1; ELSE s1:=0; END_IF; END_IF;
```

It is therefore specified that for each definition, **ELSE** refers to the most recent **IF**. The statement shown above is therefore interpreted as follows:

```
IF e1 THEN
  ( IF e2 THEN
    s1:=1;
  ELSE
    s1:=0;
  END_IF; )
END_IF
```

CASE statement

A **CASE** consists of an expression (the selector) and a list of branches, which can be of any length. Each of these branches is preceded by one or more constants or the keyword **ELSE**. The selector must be an integer data type.



Constants must not be defined more than once and must also conform to an integer data type that is compatible with the selector type. A branch that is preceded by a constant is carried out if the value of the constant is equal to that of the selector. The same applies if a range includes the value of the selector. If the value of the selector does not agree with either a constant or a range, the branch following **ELSE** is carried out. If no **ELSE** branch is defined, the program continues with the next statement following the **CASE** statement.

A few examples:

```
CONST
  plus:= 1;
  minus:= 2;
  times:= 3;
END_CONST
CASE operator OF
  plus: a := b + c;
  minus: a := b - c;
  times: a := a * b;
END_CASE;
```

```
CASE state OF
  0: display_text := 'O.K.';
  1,5: display_text := 'Excessive temperature';
  2 .. 4: display_text := 'Torque';
  7: display_text := 'No feedback';
  6, 8 .. 10: display_text := 'No auxiliary power';
  ELSE display_text := 'Unknown error';
END_CASE;
```

Loops

Loops specify parts of programs that are repeated. If the number of repetitions is known in advance, use of the **FOR** statement is recommended. If not, **WHILE** or **REPEAT** should be used.

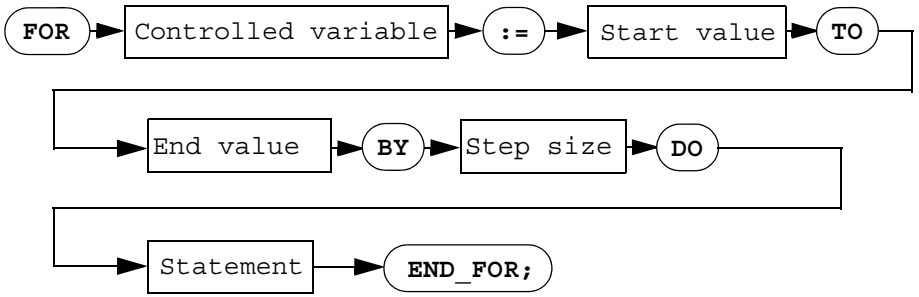
FOR statement

The **FOR** statement carries out a loop in which new values are assigned to a variable (the controlled variable). The controlled variable must be an integer data type.

The definition of a loop with **FOR** includes identification of a starting and ending value and a step size. The starting value, step size and ending value are expressions. These expressions must be of an integer data type that is compatible with the type of the controlled variables. Identification of the step size with **BY** is optional. If the step size is not explicitly stated the loop is carried out with step size 1.

At the start of the loop the controlled variable is set to the starting value and each time the loop is run through it changes by the step size until the ending value is reached or exceeded. The expressions for the step size and ending value are calculated once at the start of the loop and stored as constants for subsequent use in the loop.

Every time the loop is run through the statement in the body of the loop is carried out once.



After the loop is completed the controlled variable has the value *ending value + step size*.

This value must not exceed the maximum value that is defined by the data type of the controlled variable.

A couple of examples:

```
Maximum := MAX_VAL;
FOR lw := 2 TO 63 DO
  IF Data[lw] > Maximum THEN
    Maximum := Data[lw];
    MaxIdx := lw;

  END_IF;
END_FOR;
```

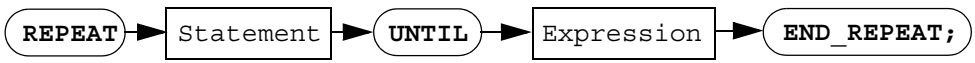
```
Sum := 0;
FOR i := 10 TO 1 BY -1 DO
  FOR j := 1 TO 2 DO
    IF FLag THEN EXIT; END_IF;
    Sum := Sum + j;
  END_FOR;
  Sum := Sum + i;
END_FOR;
```

If the ending condition of a loop is already satisfied before it runs the first time (i.e. ending value < starting value), the loop and all the statements it contains are skipped. The following loop is skipped:

```
FOR lw := 2 TO 1 DO
  s1:=5;
END_FOR;
```

REPEAT statement

The **REPEAT** statement contains an expression whose truth value determines whether it is to be repeated by the **REPEAT UNTIL** block that is included. The result of the expression must be of the data type BOOL.



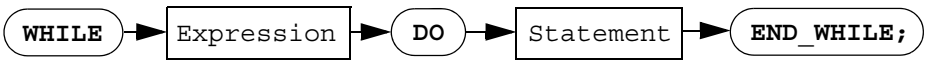
The statement is carried out repeatedly until the expression takes on the value **TRUE**. The statement is carried out at least once because the expression is not evaluated until **UNTIL** is reached. A couple of examples of the **REPEAT** statement:

```
j := -4;
REPEAT
  j := j + 2;
UNTIL j > 60
END_REPEAT;
```

```
REPEAT
  a := in1 + in2;
  b := 2 * in1;
  c := in1 * in2;
UNTIL EndCondition
END_REPEAT;
```

WHILE statement

The **WHILE** statement contains an expression whose truth value determines whether or not the statement that follows **DO** is to be carried out again. The result of the expression must be of the data type BOOL.



The expression is evaluated each time before the part of the program to be repeated is carried out. If the value **FALSE** is returned before the statement is carried out for the first time, the block that follows **DO** is not carried out at all. Otherwise it is repeated until the expression takes on the value **FALSE**.

A couple of examples:

```
WHILE i > 0 DO
  i := i - in1;
  lw := lw+ 1;
END_WHILE;
```

```
i := StartValue;
WHILE Data[i] <> x DO
  i := i + 1;
END_WHILE;
Index := i;
```

The essential difference between **WHILE** and **REPEAT** is that **WHILE** repeats a loop for as long as the expression returns **TRUE**. **REPEAT** repeats the loop until the expression takes on the value **TRUE**. Unlike **WHILE**, a **REPEAT** statement is run through at least once.

Control statements

The statement **EXIT** offers the option of exiting a loop before the ending condition is reached. If the **EXIT** statement is inside nested loops, only the loop level concerned is exited with **EXIT**, e.g.

```
WHILE i > 0 DO
  lw := lw+ 1;
  IF lw > MAX_LW THEN
```

```
    EXIT;
  END_IF;
  i := i - in1;
END_WHILE;
```

After the **EXIT** statement, the program continues with the statement that follows **END_WHILE**.

The statement **RETURN** causes an ST program to be prematurely abandoned. **RETURN** interrupts the processing of the current program and control of the program flow is transferred to the next level up. That can be:

- the program list which called an ST program or
- the program that called a user-defined function block.

RETURN is used first and foremost in user-defined function blocks, e.g.

```
CASE error_state OF
  2 .. 4: OUT := hold_value;
  5: OUT := 0.0; RETURN;
  6, 8 .. 10: OUT := fix_value;
  ELSE OUT := 0.0;
END_CASE;
```

8.3.6 Limits of the system

Local elements

The number of local elements is limited to 65526 for each ST program. Local elements are the local variables, every element of a structured variable and every individual array element and intermediate store within the program.

Programming of loops

Every ST program is processed in the context of a user task. In other words, the ST program is carried out once in the space of the task cycle time.

When loops are used, a part of the ST program, the statements within the loops, is run through many times in a task cycle. This leads to increased runtime for the user task. The time spent executing a loop should not exceed 5 ms.

When programming loops, there is a risk of creating endless loops. The following loop is not ended:

```
REPEAT
  i := i +1;
  sEnd := FALSE;
UNTIL sEnd
END_REPEAT;
```

This loop is repeated continuously with the result that none of the other programs in the user task can be processed.

Memory occupancy

Structured text is a high-level language that is translated into machine code by code generation. The complex statements mean that the machine code produced is substantially longer than the source text.

In multidimensional arrays the number of individual elements increases very rapidly. The array
`ARRAY 1 .. 100, 1 .. 100] OF REAL`
contains 10,000 elements and requires approximately 39 K bytes of storage space.

User-defined function blocks

Names of inputs and outputs

User-defined function blocks with identical input and output names cannot be used in ST programs. During class definition these blocks have acquired input and output names with more than 3 characters. The first three characters of some inputs and/or outputs are identical.

In ST programs, the inputs and outputs of function blocks are accessed by their names, not their position on the block. There is no unambiguous supply to inputs or collection from outputs for identical names of a user-defined function block.

Class names

Class names of user-defined function blocks must not contain any special ST symbols

+ - * / & = < > [ ] . , ( ) : ; ’ @ # $

These user-defined function blocks cannot be used in ST programs.

Since the class name of the user-defined function block is interpreted in ST as a identifier, the use of special symbols is not allowed.

8.3.7 Examples

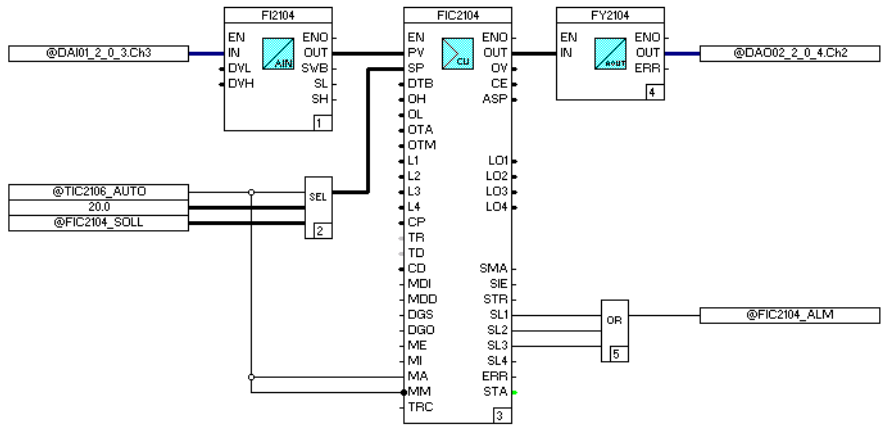
Simple control loop

The following example shows the programming of a simple control loop in ST. The I/O signals are accessed using component names. The external set point is set as a function of the global variable TIC2106_AUTO, which also determines the operating mode. The global variables and I/O components are accessed via the process image.

```
PROGRAM control loop
CONST
  (* Definition of constants *)
  MAN_SP := 20.0;
END_CONST
VAR
  (* declaration of function blocks *)
  FI2104: AI_TR;
  FIC2104: C_CU;
  FY2104: AO_TR;
  (* declaration of local variables *)
  rLocalVar1, rLocalVar2: REAL;
END_VAR
VAR_EXTERNAL
  (* declaration of HW objects for I/O access *)
  DAI01_2_0_3: DAI01;
  DAO02_2_0_4: DAO01;
  (* declaration of global variables *)
  FIC2104_SETP: REAL;
  TIC2106_AUTO: BOOL;
  FIC2104_ALM: BOOL;
END_VAR
```

```
  (* call input transformation *)
  FI2104(IN:= @DAI01_2_0_3.Ch3, OUT=> rLocalVar1);
  (* assign external set point *)
  IF @TIC2106_AUTO THEN
    FIC2104.SP := @FIC2104_SETP;
  ELSE
    FIC2104.SP := MAN_SP;
  END_IF;
  (* assign operating mode *)
  FIC2104.MA := @TIC2106_AUTO;
  FIC2104.MM := NOT(@TIC2106_AUTO);
  (* call control function blocks *)
  FIC2104(PV:= rLocalVar1, OUT=> rLocalVar2);
  (* further use of outputs *)
  @FIC2104_ALM := FIC2104.SL1 OR FIC2104.SL2 OR FIC2104.SL3;
  (* call output transformation *)
  FY2104(IN:= rLocalVar2, OUT=> @DAO02_2_0_4.Ch2);
END_PROGRAM
```

This ST program carries out the same functionality as the following FBD program:



to005.bmp

Linearization

The following is an example of linearization. The array *Curve* is initialized at the declaration position. A **FOR** loop is used to search through the array for the

appropriate input range. When the condition is satisfied the loop is terminated after calculation of the output value.

```
PROGRAM lin
VAR
  (* declaration of local variables *)
  Curve : ARRAY [1..10, 1..2] OF REAL :=
          0.0,    0.0,    (* X1, Y1 *)
          8.0,    12.0,    (* X2, Y2 *)
          13.0,    24.4,    (* X3, Y3 *)
          32.6,    28.4,    (* X4, Y4 *)
          47.0,    16.0,    (* X5, Y5 *)
          62.0,    58.0,    (* X6, Y6 *)
          78.4,    82.0,    (* X7, Y7 *)
          83.5,    85.0,    (* X8, Y8 *)
          92.0,    63.7,    (* X9, Y9 *)
          100.0,   100.0;    (* X10, Y10 *)
  Index : INT;
END_VAR
VAR_EXTERNAL
  (* declaration of global variables *)
  IN: REAL;
  OUT: REAL;
END_VAR

  (* start of linearization *)
  FOR Index:=1 TO 10 DO
    IF IN <= Curve[Index,1] THEN       (* range found*)
      IF Index = 1 THEN                (* low limit *)
        OUT := Curve[Index,2];
        EXIT;                       (* exit FOR loop *)
      END_IF;
  (* calculate output from straight line equation *)
  (* Y := (((Y2 - Y1) / (X2 - X1)) * (X - X1)) + Y1; *)
      OUT := (((Curve[Index,2] - Curve[Index-1,2]) /
               (Curve[Index,1] - Curve[Index-1,1])) *
              (IN -Curve[Index-1,1])) + Curve[Index-1,2];
      EXIT;                         (* exit FOR loop *)
```

```
    ELSE
      IF Index = 10 THEN                (* high limit *)
        OUT := Curve[Index,2];
        EXIT;
      END_IF;
    END_IF;
  END_FOR;
END_PROGRAM
```

MIN_MAX - user-defined function block

In its running time the user-defined function block MIN_MAX determines the maximum and the minimum of the sampled input signal IN. The maximum and minimum can be reset via the RES input.

```
FUNCTION_BLOCK MIN_MAX

(* the following declarations originate from the interface- *)
(* definition of the user-defined function block *)
(* and cannot be altered in the ST program. *)
  VAR_INPUT
  IN : REAL;
  RES : BOOL;
  END_VAR
  VAR_OUTPUT
  MAX : REAL;
  MIN : REAL;
  END_VAR
  VAR (* VAR_DPS *)
  END_VAR
  VAR (* PARA_DPS *)
  END_VAR
  (* PARA_VIS
    ClassName : TEXT;
    TagName : TEXT;
    ShortText : TEXT;
    LongText : TEXT;
    SelState : BOOL;
```

```
  END_VAR *)
(* End of interface definition *)

(* declarations *)
CONST
  MAX_REAL := 1.0e38;
END_CONST
VAR
  intMax : REAL := -MAX_REAL;
  intMin : REAL := MAX_REAL;
END_VAR
(* program *)
IF RES THEN
  (* RES input is set *)
  MAX := 0.0;
  MIN := 0.0;
  intMax := -MAX_REAL;
  intMin := MAX_REAL;
ELSE
  (* normal function *)
  intMax := Max(IN, intMax);
  intMin := Min(IN, intMin);
  MAX := intMax;
  MIN := intMin;
END_IF;
END_FUNCTION_BLOCK
```

8.4 Edit an ST program

8.4.1 Insert ST elements

To simplify the processing of ST programs, ST elements with the basic syntax can be inserted directly into the text.

> **Elements** > **Insert ST Syntax** > select element to be inserted

or

> Context menu > **Insert ST Syntax** > select element to be inserted.

or

In **Explorer** pane:
> **Libraries** tab > **ST Elements** > select element to be inserted.

An **IF** statement is inserted into the ST program in the following form:

```
IF Expression THEN
  ThenStatementList;
ELSE
  ElseStatementList;
END_IF;
```

For the following ST elements the syntax can be inserted into the program:

- VAR
- VAR_EXTERNAL
- CONST
- Date & Time
- Time
- IF
- CASE
- FOR
- WHILE
- REPEAT
- RETURN
- EXIT
- Assignment

8.4.2 Insert variables and function blocks

Before variables and function blocks are used in ST programs they must be declared, i.e. made known to the ST program. Variables and function blocks can be directly entered as text. This does not enter them in the variable list or the tag list. Entering them there is done via instantiate. Functions can be used in ST programs without declaration.

Variables that are already available can be used directly in an ST program.

Insert variable

Within an ST program, variables that have already been defined can be inserted directly. If the cursor is in the **`VAR_EXTERNAL END_VAR`** block a variable declaration is inserted. The variable name is inserted in the program code for subsequent use in the program:

> Elements > Select global variable or **F2** key

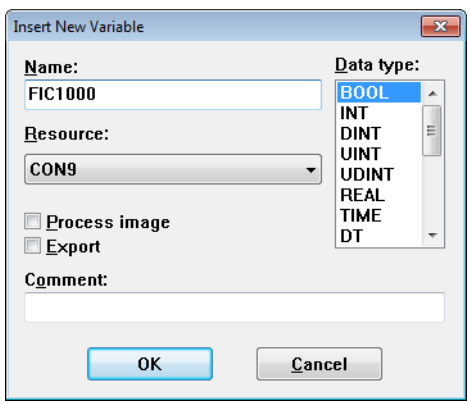
> Select an existing variable in the project from the list

New variables can also be created directly in an ST program:

> Position the cursor in the **`VAR_EXTERNAL END_VAR`** block

> Elements > Create global variable

The following dialog appears for creating a new variable in the variable list:



di0133us.png

Name name of the new variable.

Data type Identify the data type of the newly defined variable. The standard data types and all user-defined data types can be selected from the list.

Resource Specify the assignment of the variable to the resource. Every variable is assigned to exactly one resource. Access to this variable by other resources is read-only.

Variable via process image

☑ Access via the process image is preset for all uses of the variables.

Export ☑ Variable is enabled for reading by other resources.

Comment Any desired text can be added to a variable for clarification.

After definition of the variables is terminated this is automatically adopted in the system-wide variable list and can be used in other programs (see Section 1, Variables).

New variables can also be declared as text. The data type must also be entered as text, e.g.

```
VAR_EXTERNAL
  TI203: REAL;
  TI203_MODE: BOOL;
END_VAR
```

With text declaration, the variable is not yet entered in the list of variables. The variable still has to be instantiated for this to take place. See also Global variables on page 241.

Insert function blocks

Function blocks can be declared directly within a **VAR END_VAR** block.

User-defined function blocks with special symbols in class names cannot be used in ST programs. See Limits of the system on page 257.

> Position the cursor in the **VAR END_VAR** block.

> Select the function block either from the **Elements** > **Blocks** menu or from the **Explorer pane** > **Libraries** tab

The ST editor opens the dialog.

> Enter a new valid tag name in the dialog.

> If desired, carry out further parameterization.

> Click **OK** to close the dialog

Outside a **VAR END_VAR** block, the following syntax is entered into the ST program, e.g.

```
<TagName> : LIN;
```

This text can be moved into an **VAR END_VAR** block.

Function blocks can only be declared within a **VAR END_VAR** block.

New function blocks can also be declared as text. The function block type must also be entered as text, e.g.

```
VAR
  TI203: AI_TR;
END_VAR
```

With text declaration, the function block is not yet entered in the tag list. It still has to be instantiated for this to take place. See also Function blocks on page 244.

Instantiate

After variables and function blocks have been textually defined (declared) they still do not appear in the corresponding lists (variable list and tag list). A further step, **Instantiate**, is needed for this.

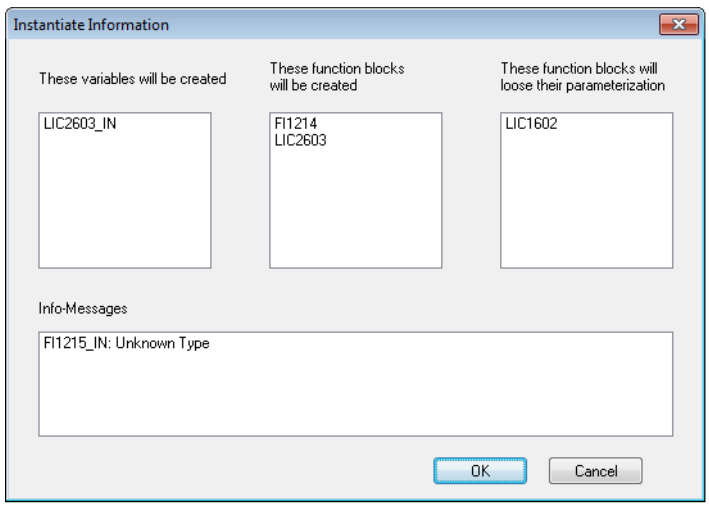
> Elements > Instantiate

or

> Editor menu bar >click **Instantiate**

Instantiate links the parameter data for the function blocks to the ST program. After the function block declaration has been deleted the parameter data is still stored in the ST program and can be erased by instantiate.

The dialog summarizes the objects to be instantiated and/or those no longer required:



to006us.png

These variables will be created

The variables in the field will be entered in the variable list.

These blocks will be created

The function blocks in the field will be entered in the tag list.

These blocks will lose their parameterization

These function blocks have been deleted from the ST program. The parameter settings for these function blocks are still included in the ST program and will be deleted when the instantiate is confirmed.

Info-Message This field displays information on objects that cannot be instantiated.

OK Instantiate is carried out and the parameters for function blocks that are no longer needed are deleted

8.4.3 Working with variables

Before variables are used in ST programs they must be declared, i.e. made known to the ST program. Information on inserting variables is given in Insert variables and function blocks on page 265.

Access to variables

There is read access and write access to variables. Variables to the left of the assignment operator have write access. All applications to the right of the assignment operator are read-only access, e.g.

```
var4 := var1 + SIN(var2 * var3);
(* var4 can be written to *)
(* var1, var2 and var3 are read from *)
```

Multiple write access to the same variable is allowed in an ST program and does not lead to errors, e.g.

```
var1 := 5:
var2 := 3 * var1;
var1 := 42;
```

Process image

As with other program editors, access to a variable in ST programs can be direct or via the process image. Access is gained via the process image by placing @ in front of the variable name, e.g.

```
var1 := @var4 + var2;
(* var1 can be written to directly *)
(* var4 is read via the process image and var2 is read
directly *)

@var3 := var4 + var5;
(* var3 is written to via the process image *)
(* var4 and var5 are read directly *)
```

If a variable is read many times in an statement, the same type of access should be used every time. In the statement

```
var1 := var2 + SIN(@var2);
```

inconsistent data supply of var2 may occur, as var2 is read from two different places (directly and via the process image).

Using system variables

System variables are declared by definition. Explicit declaration in the ST program is not necessary. System variables can be used like global variables. Access via the process image is possible, e.g.

```
load := @ps12.CPU_Load;
date := @ps12.DateTime;
```

8.4.4 Working with functions

Using functions

Functions can be used directly in ST programs without a declaration.

> **Elements** > **Blocks** > select function from library

or

In Explorer pane:
> **Libraries** tab > select elements from a category to be inserted

Functions can also be inserted directly in ST programs by giving the name. Refer to ***Engineering Manual, Functions and Function Blocks***.

Values are transferred in a bracketed argument list. The individual arguments are separated by commas:

Function name(Argument_1, Argument_2, .. , Argument_n)

It is also possible to call a function within the argument transfer of another function, e.g.

```
CONST
  AllChar :='0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ';
END_CONST
```

```
VAR
  B1, B2: BYTE;
  S1, S2: STR8;
END_VAR

B1 := EXTCT(1, TO_WO(S_FIND(AllChar, S_LEFT(S1, 1)) + 48));
(* changes a 1 character STR8 into a BYTE *)
S2 := TO_STR8(S_MID(AllChar, 1, TO_IN(PBYWO(B2))-48));
(* changes a BYTE into a 1 character STR8 *)
```

Data type of functions

In ST programs the data types of functions are automatically adapted to suit the transferred arguments. There is no need for the data type to be adapted explicitly to the transferred arguments. If necessary, the data type of the argument can be changed to suit by conversion functions, e.g.

```
VAR
  i1, i2: INT
END_VAR
(* extract the root from an INT value *)
i2 := to_in(SQRT(to_re(i1)));
```

The possible data types for a function must be taken from the relevant function description. See ***Engineering Manual, Functions and Function Blocks***.

If constants are transferred exclusively to a polymorphic function (with different data types) as the argument, at least one explicit type change is needed, e.g.

```
CONST
  CONST_VAL := 12;
END_CONST
VAR
  i1: INT;
END_VAR
i1 := add(to_in(CONST_VAL), 42);
```

Number of function inputs

In ST programs the number of function inputs is determined by the argument list, eg.

```
VAR
  b1, b2, b3, b4, b5, b6: BOOL
END_VAR
b1 := AND(b2, b3, b4);
b1 := OR(b2, b4, b5, b6);
```

The number of arguments must not exceed the maximum input number. It is not possible to set the input number explicitly in ST programs.

8.4.5 Working with function blocks

Before function blocks are used in ST programs they must be declared, i.e. made known to the ST program. Information on inserting function blocks is given in Insert function blocks on page 266.

Position function blocks in the program

Function blocks in ST programs are called in statements (see Statements on page 247). The tag name of the function block must be used in the statement to call it, e.g.

```
VAR
  TI104: LIN;     (* Declaration of the function block*)
  in, out: REAL; (* Declaration of local variables *)
END_VAR


(* calling the function block *)
TI104(IN := in, OUT => out);
```

Function blocks may only be called once in the ST program.

Use function blocks in loops

Certain function blocks need uniform sampling to fulfil their functionality, i.e. they must be calculated at uniform intervals. Controller blocks are examples of these. They must not be called within loops.

Supply function block pins

A distinction is made between mandatory pins and optional pins for block inputs and outputs. The two types of pin are represented in the same way in ST programs.

User-defined function blocks with identical input and output names cannot be used in ST programs. See Limits of the system on page 257.

Supply to and retrieval from mandatory pins must take place with the block call. All mandatory pins can be inserted directly behind the function block call.

Position the cursor on the function block call (tag name)

> **Elements > insert mandatory parameter**

A comma-delimited argument list with all mandatory pins is created, e.g.

```
TI104(IN:= (*REAL*), OUT=> (*REAL*));
```

The data type is given for each pin in the form of a comment. Inputs are supplied via the assignment operator :=. They can be supplied directly with constants. Outputs are written to variables with =>, e.g.

```
TI104(IN:= 45.0 (*REAL*), OUT=> out (*REAL*));
```

Supply to and retrieval from optional pins can take place with the block call or at any point in the ST program. They can be inserted directly at the function block call.

> Position the cursor on the function block call (tag name)

> **Elements > Insert one parameter**

> Select input or output from the list

(Inputs and outputs are separated by a horizontal line).

An additional pin is appended to the comma-delimited argument list, e.g.

```
TI104(IN:= (*REAL*), OUT=> (*REAL*), STA=> (*INT*));
```

Inputs and outputs that support more than one data type have the comment text `(*Other*)`, e.g.

```
VAR
  TIR104: TREND;
```

```
END_VAR
TIR104(IN1:= (*Other*),IN2:= (*Other*),IN3:= (*Other*));
```

In addition to block call, optional pins are supplied via assignments. The pin name is then appended to the tag name with a dot separator, e.g.

```
(* Assignment of inputs for a PID controller *)
IF @TIC2106_AUTO THEN
  FIC2104.SP := @FIC2104_SETP;
ELSE
  FIC2104.SP := MAN_SP;
END_IF;
FIC2104.MA := @TIC2106_AUTO;
FIC2104.MM := NOT(@TIC2106_AUTO);
(* calling the controller block *)
FIC2104(PV:= @FIC2104_PV, OUT=> @FIC2104_OUT);
(* further use of outputs *)
@FIC2104_ALM := FIC2104.SL1 OR FIC2104.SL2 OR FIC2104.SL3;
```

The names of the function block pins are described in the ***Engineering Manual, Functions and Function Blocks***. If the name of a function block pin is the same as a keyword (see Special symbols and reserved words on page 234), an underline must precede the pin name, e.g.

```
VAR
  pin_dt: P_DT;
END_VAR
pin_dt(DAY := 18, MON := 6; YEA:= 2002, _DT=> dt1);
```

Function block outputs cannot be written to.

If an input is supplied before the block call and in the block call, the supply in the block call has precedence, e.g.

```
VAR
  ana_mon: M_ANA;
END_VAR
ana_mon.L1 := 12.0;
ana_mon(IN := In12, L1 := 42.0);
```

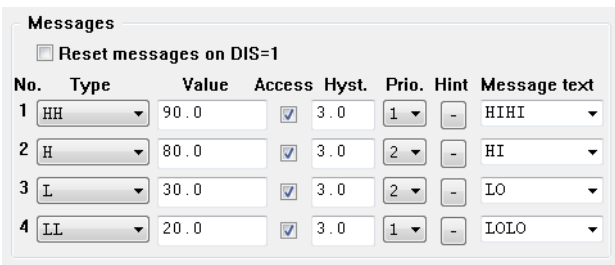
The limit value L1 in the function block has the value 42.0.

If the value of an output is used before the block call, the initial value or the value from the most recent calculation cycle is transferred.

Function block pin and parameters

In certain function blocks it is possible to select use as an input (pin) or as a parameter, e.g. the limit value in the M_ANA for individual inputs. Function block inputs can also be used in ST programs outside the function block call. For this reason it has been laid down that in cases of simultaneous use as an input and as a parameter, the input has priority. If both the input and the parameter are used the ST program issues a plausibility warning.

Configuration example:



to014us.png

```
VAR
  pin_ana: M_ANA;
END_VAR
pin_ana.L1 := Limit1;
pin_ana.L2 := Limit2;
pin_ana(IN := Input);
```

The limit values are assigned as follows:

Limit value 1	Only the input is configured. The limit present at input L1 is used as the limit value.
Limit value 2	Input and parameter are configured. The limit present at input L2 is used as the limit value.
Limit value 3	Only the parameter is configured and the parameter value is used as the limit value.
Limit value 4	Not configured.

Inputs that can also be configured as parameters have priority over the parameter in the ST program.

Negate function block pins

The values of the Boolean inputs and outputs of a function block can be inverted at every point of use. The complement operator NOT must be used to do this, e.g.

```
FIC2104.MM := NOT(@TIC2106_AUTO);
```

All function blocks have non-negated input and output pins as default.

The function block terminal to be inverted must be of the data type BOOL (i.e. binary).

Parameterize function blocks

> Select the tag name of the function block to be parameterized.

> **Edit** > **Parameters**

or

> Double-click the tag name of the function block

The first parameter definition dialog of the function block is called. The function block can be parameterized as described in Handling the parameter dialogs on page 130.

Check function blocks

In ST programs it is not possible to check function blocks for plausibility from the parameter dialog. Function blocks may only be checked in the context of the ST program.

> **Editor** > **Check**

or

> Editor menu > **Check editor**

Activation from the error list takes place as in other program editors.

8.4.6 Program user-defined function blocks

A user-defined function block is created as described in Section 10, User Function Blocks. If structured text is used as the program, certain special features have to be observed (see also Limits of the system on page 257).

An ST program is enclosed between the keywords **FUNCTION_BLOCK** and **END_FUNCTION_BLOCK**.

```
FUNCTION_BLOCK STufb1
;
END_FUNCTION_BLOCK
```

Interface definition

The interface definition of the user-defined function block (see Interface editor on page 356) is adopted as a non-editable (grey) block in the ST program. The interface cannot be modified in the ST program, this must always be done in the interface editor. All variables defined in the interface that are loaded onto the process station can be used in the ST program. See also Inputs and outputs on page 243.

```
FUNCTION_BLOCK STufb_P

VAR_INPUT
    IN : REAL;
END_VAR

VAR_OUTPUT
    OUT : REAL;
END_VAR

VAR (* VAR_DPS *)
END_VAR

VAR (* PARA_DPS *)
END_VAR

(*
PARA_VIS
    ClassName : TEXT;
    TagName : TEXT;
    ShortText : TEXT;
    LongText : TEXT;
    SelState : BOOL;
END_VAR
*)

;
END_FUNCTION_BLOCK
```

to013.bmp

Local variables can be declared in the ST program. There is no access to local variables from the faceplate.

The keyword **VAR_EXTERNAL** for declaring global variables must not be used in user-defined function blocks.

Blocks in the UFB

Function blocks that are used in user-defined function blocks must be declared in a **VAR END_VAR** block. This ensures that they always have a name in the class definition.

In the instance (Zoom to user FB) every embedded function block can be given an individual name.

> Open parameter dialog for the function block

> Enter tag name

This individual tag name is not displayed in the ST program.

8.5 General processing functions

8.5.1 Bookmarks

Bookmarks are used to identify individual lines in an ST program for rapid selection. Bookmarks are displayed as a light blue rectangle in the mark column.

```
FOR Index:=1 TO 10 DO
  IF @IN <= Curve[Index,1] THEN      (* Sector found *)
    IF Index = 1 THEN               (* low limit *)
      @OUT := Curve[Index,2];
      EXIT;                    (* Exit FOR loop *)
    END_IF;
(* Calculation of output according to the straight line equation *)
(* Y := (((Y2 - Y1) / (X2 - X1)) * (X - X1)) + Y1; *)
    OUT := (((Curve[Index,2] - Curve[Index-1,2]) /
             (Curve[Index,1] - Curve[Index-1,1])) *
            (IN -Curve[Index-1,1])) + Curve[Index-1,2];
    EXIT;                        (* Exit FOR loop *)
  ELSE
    IF Index = 10 THEN              (* high limit *)
      @OUT := Curve[Index,2];
      EXIT;
    END_IF;
  END_IF;
END_FOR;
```

to007us.bmp

Bookmarks can be switched on and off. If there is no bookmark in the line the switch sets it, if there is one it deletes it.

> Position cursor in desired line

> **Edit > Toggle bookmark**

or

> **CTRL + F7** key

You can jump to a bookmark from any point in the ST program. Thereafter it jumps to the next bookmark, and so on:

To jump forward (towards the end of the program):

> **Edit > Goto next bookmark**

or

> **F7** key

To jump backwards (towards the beginning of the program):

> **Edit** > **Goto previous bookmark**

or

> **SHIFT + F7** key

8.5.2 Breakpoints

Breakpoints are used for tracing faults in programs. See Section 11, Debugger are displayed as a brown circle in the mark column.

```
FOR Index:=1 TO 10 DO
  IF @IN <= Curve[Index,1] THEN      (* Sector found *)
    IF Index = 1 THEN                (* low limit *)
      @OUT := Curve[Index,2];
      EXIT;                      (* Exit FOR loop *)
    END_IF;
(* Calculation of output according to the straight line equation *)
(* Y := (((Y2 - Y1) / (X2 - X1)) * (X - X1)) + Y1; *)
    OUT := (((Curve[Index,2] - Curve[Index-1,2]) /
             (Curve[Index,1] - Curve[Index-1,1])) *
            (IN -Curve[Index-1,1])) + Curve[Index-1,2];
    EXIT;                        (* Exit FOR loop *)
  ELSE
    IF Index = 10 THEN               (* high limit *)
      @OUT := Curve[Index,2];
      EXIT;
    END_IF;
  END_IF;
END_FOR;
```

to008us.bmp

Breakpoints can be switched on and off. If there is no breakpoint in the line the switch sets it, if there is one it deletes it.

> Position the cursor on the desired line

> **Edit** > **Toggle breakpoint**

or

> **F9** key

Breakpoints that have been set can be disabled and enabled

> Position the cursor on the desired line > context menu

> **Disable breakpoint / Enable breakpoint**

8.5.3 Find and replace

In ST programs, any text can be found. The options **Find**, **Find next** and **Replace** are used to search or find text in the ST program.

> **Edit** > **Find**

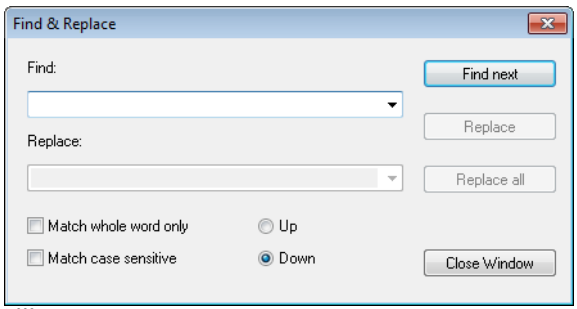
> **Edit** > **Find next**

> **Edit** > **Replace**

or

> **F3** key

Find and replace starts at the current cursor position. At the end of the ST program the find continues automatically from the program start and vice versa. If a text area is selected when the **Find & Replace** dialog box is opened, this text area is shown as the default in the **Find** field.



to009us.png

Find The term entered is searched for in the ST program. Every find term is saved in a list and can be recalled from the list later.

Replace Replaces the find term with the term that has been entered. Every replacement term is saved in a list and can be recalled from the list later. If the field is empty, the find term is deleted.

Match whole words only

☑ The find term is looked for as a whole word. If the find term is part of another word it is ignored. Thus, for example, the find term linear is not found in the word linearization.

Match case sensitive

☑ The find only finds words with upper and lower case exactly as in the find term. Only exact matches are found.

☐ Differences of case are ignored.

Up ◉ The find proceeds from the current position towards the beginning of the program

Down ◉ The find proceeds from the current position towards the end of the program

Find next The next occurrence of the find term is found according to the settings in the ST program.

Replace The find term is replaced with the term in the Replace field.

Replace all All occurrences of the find term in the ST program are replaced by the term in the Replace field without prior confirmation.

Close window Ends the search and closes the Find & Replace dialog box.

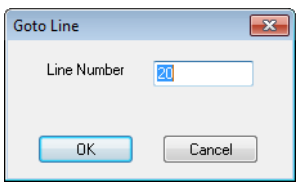
The **Replace** field is disabled in the **Find & Replace** dialog box when **Find** and **Find next** options are selected.

8.5.4 Goto line

In ST programs it is possible to jump to any line.

> Edit > Goto line

The current line is offered when the dialog is called.



to010us.png

Line number Enter the number of the line to jump to.

The current line number is displayed in the state line.

8.5.5 Block operations

Select program elements

Select individual program elements

Select by placing the cursor on the desired program element and clicking the left button.

The selection area is the whole text of the program element (e.g. variable name or tag name). The selection of individual program elements is not highlighted in the ST program.

After the selection it is possible to open e.g. the parameter dialogs of function blocks

Mark areas of text

> Hold down the left mouse button > mark an area of text

```
OUT := (((Curve[Index,2] - Curve[Index-1,2]) /
        (Curve[Index,1] - Curve[Index-1,1])) *
        (IN -Curve[Index-1,1])) + Curve[Index-1,2];
```

to011.bmp

Inside a line, characters are marked. Otherwise complete lines are marked. After marking, the desired operation can now be carried out. Example: > **Edit** > **Cut**

Extending an area of marked text

> Press **SHIFT** key and hold it down > select more characters or lines

The additional characters or lines are included in the marked area.

Deselect program elements

Deselect text area

> Click unmarked point in ST program

The text area is deselected and shown as such.

A selection is automatically cancelled by opening a different window.

Reducing an area of marked text

> Press **SHIFT** and hold it down > position cursor within the marked text

The marking is reduced to coincide with the selected cursor position.

Copy

> **Edit** > **Copy**

or

> **CTRL + C**

Before an area of text can be copied it must be marked. Copying places the marked text into an internal store. Areas of text that have been placed in the internal store by a previous copying operation are overwritten. To find out whether an area of text is in the internal store, look at menu item **Insert** in the **Edit** menu or in the context menu. If the menu item is inactive, the internal store is empty.

Marked areas of text can also be copied into other ST programs.

Cut and delete

> **Edit > Cut**

or

> **Delete** or **CTRL + X** or **DEL**

If areas of marked text have been cut, they can be put back into the program again using **Paste**. When text is cut, areas of text already present in the internal store are overwritten.

If areas of text are deleted they can only be inserted again immediately afterwards with **Undo**, they cannot be re-inserted later. Areas of deleted text can only be restored by exiting the program without saving.

When function blocks are cut the associated parameter data is not transferred to the internal store but stored in the ST program. When a function block is pasted into the same ST program the parameter data is retained but when the block is pasted into a different one it loses its parameter data.

Paste

The following methods of pasting areas of text that have previously been copied or cut are available:

> **Edit > Paste**

or

> **CTRL + V**

The copied or cut area of text is inserted at the current cursor position.

Write file

A marked area of text can be written to a file for exchange with other projects and other editors.

> **Edit > Write file**

A Unicode text file with the extension .txt, which can be edited in any Unicode-enabled text editor, is created. The parameter data associated with function blocks is not written to the text file. The file thus created can be used for program exchange with other ST editors.

Read file

An ST program that has been created with another text editor can be read into the ST editor.

> **Edit > Read file**

The text file to be read in can be in Unicode or ASCII format. The file format is automatically recognized when the file is read in. The contents of the text file are inserted at the current cursor position.

Export block

A marked area of text in an ST program can be exported to a file.

> **Edit > Export block**

A file is created in a Freelance Engineering specific format. This includes the parameter data associated with a function block. The complete tag name of the function block declaration must be included in the area of marked text if the parameter data is to be exported.

Import block

A block that has been exported from an ST program can be imported.

> **Edit > Import block**

The contents of the imported file are inserted at the current cursor position. Function blocks contained in the file are imported, together with their parameter data. If the tag name already exists, a new tag name is created by appending a serial number.

The function blocks are automatically instantiated, i.e. the new tag names are entered in the tag list. Imported parameter data is assigned to the function blocks.

Undo an action

> **Edit > Undo**

This function allows the last action taken to be undone. Irrespective of this, the state of the program remains **implausible** until the next plausibility check.

8.5.6 Cross references

The cross references can be selected directly from the ST editor:

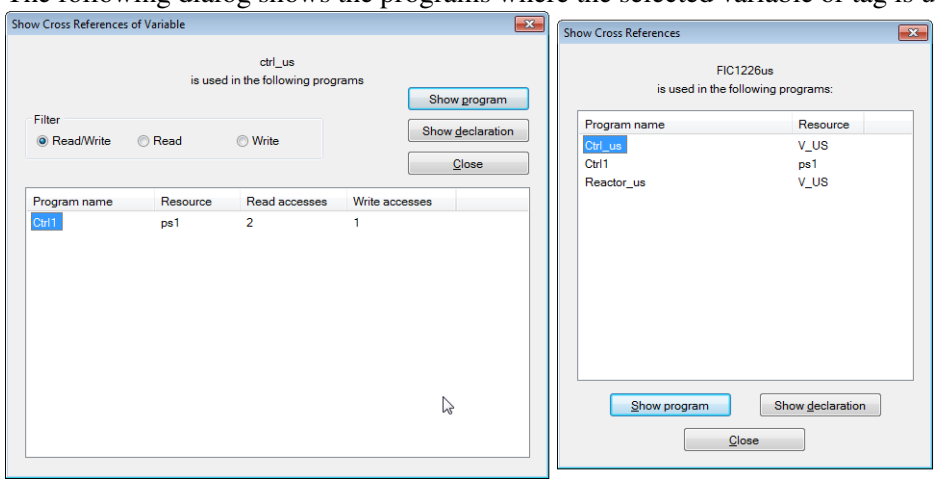
Select a variable, I/O component or tag

> **Edit > Cross reference**

or

> **F5** key

The following dialog shows the programs where the selected variable or tag is used.



CrossRef_us.png

In contrast to the variables, for the tags no read or write access is defined.

Show program

For a variable:
Call a program with prior selection of these variables, or call the module with prior selection of the I/O component.

For a tag:
Call the program with prior selection of this tag, or call the module in the hardware structure.

Show declaration

For a tag, the tag list is called, for a variable the variable list is called. If an I/O component is used directly in the program, the I/O editor of this component is opened.

Filter A filter enables only those variables to be displayed for which read-only access or write-only access exists in the programs concerned.

After activation it is possible to branch to the programs listed as cross-references.

Show next / previous cross reference

> Select a variable > **Edit** > **Cross reference** > **Find next** or **Find previous**

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

8.5.7 Program administration functions

Save the program

> **Project** > **Save Tab**

The program is saved without exiting. Programs that are not correct can also be saved and then completed at any time.

If the project is not saved in the project tree on closing or before, changes made to the program are ineffective.

Document the program

> Project > Documentation

The editor for documentation is opened as separate tab in the right pane. This can be closed from the close button available at the right side of the opened tab, and reopened later from the main menu.

Documentation administration is opened. This is where user specific project documentation is defined and output. For a description, see ***Engineering Manual System Configuration, Documentation***.

Program header

> Project > Header

A program-specific short comment can be added to the program documentation header, or this can be edited.

For drawing header / footer, see ***Engineering Manual System Configuration, Documentation***.

Edit program comment

> Project > Comment

A longer program-specific comment can be edited here to describe the functionality. For a description, see ***Engineering Manual System Configuration, Project manager***.

Print

> Options > Print

The contents of the screen are output to the standard printer.

Plausibility check

> **Editor > Check**

All inputs relevant to operation are checked for syntactical and contextual correctness. Errors, warnings and notes that are found are displayed in an error list. If the plausibility check detects errors, the processing state of the program is **implausible**.

The processing state of program elements that are newly entered, copied or moved is **implausible**.

This plausibility check reviews the accuracy and consistency of the program itself. To test the correctness in the project context call plausibility from the project tree. See Engineering Manual System Configuration, Project Tree, plausibility.

Error list

> **Editor > Show error list**

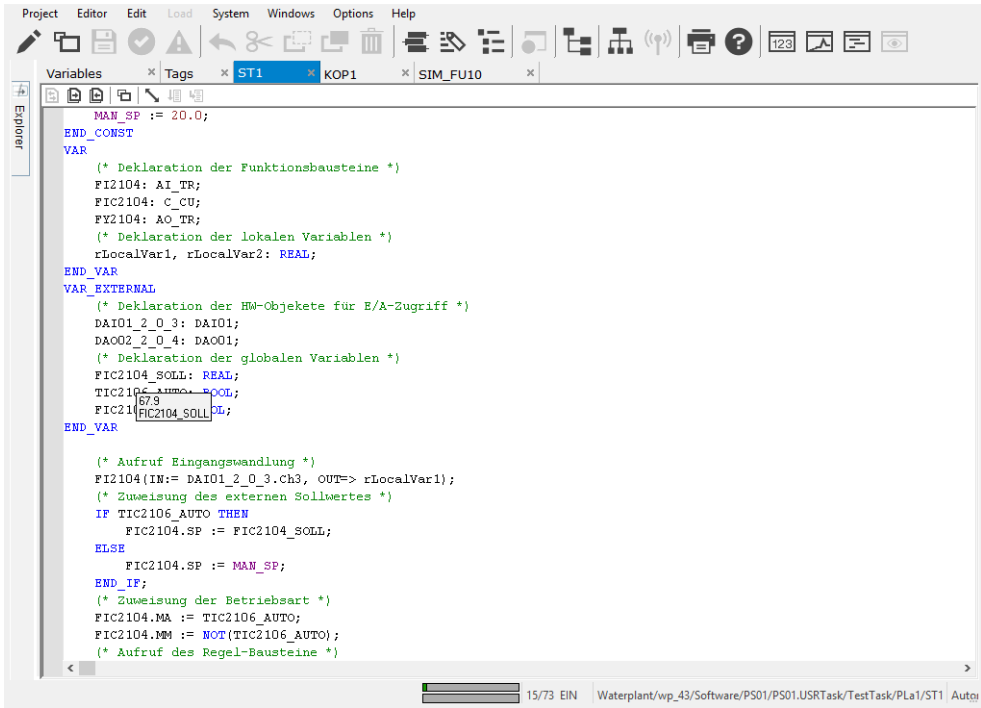
Any errors present in the program is displayed in the error list. Double-click a check message to jump to the line in the program that caused this error.

See also ***Engineering Manual System Configuration, Project tree***.

8.6 Commissioning structured text

8.6.1 User interface for commissioning

When structured text is commissioned, the program is displayed in the same way as during configuration, except that in commissioning mode the structure of the program cannot be changed.



tl027us.png

The individual function blocks can be selected and parameterized. Operating modes can also be modified from the commissioning mode.

If the program editor is opened in commissioning mode, that for showing the live value, CPU load can raise approximately up to by 15%.

8.6.2 Display of online data

Online data in the ST program

On going through the text of the ST program the currently calculated values are displayed.

If the text is found in the variable list or the local variables, the current value of the variables is displayed. If global and local variables have the same name, the value of the local variables is displayed.

If the text is a tag name, the processing state of the function block is displayed. The value of function block pins cannot be displayed in the ST program.

Moreover, values within a cycle can be described once.

> Right-click on a variable or a function block pin > **Write value** > Enter value > **OK**

Writing a value must not be confused with enforced setting on the I/O module. The value that is written can be overwritten from the program in the next cycle.

Window for online data

As with other program editors, the value window and the trend window can be used in ST programs. Local variables can be entered in both windows.

To support error tracing, ST programs have an additional window, the watch window, for online data.

> Select variable > **Windows** > **Add value to watch window**

The values in the watch window are not updated on each cycle. See also Section 11, Debugger, Watch window on page 404.

8.6.3 Error tracing

The debugger can be used for ease of error tracing in ST programs. The debugger is described in Section 11, Debugger.

9 Sequential Function Chart (SFC)

9.1 General Description – Sequential Function Chart

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 sub tasks of the user program.

The sub tasks 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), instruction list (IL), or structured text (ST). 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, where as 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 **Security Lock**, operation of the entire SFC can be assigned to individual user groups or interlocked for certain user groups.

The Freelance system offers the options of operating the SFC program in either the automatic or manual operating mode. In automatic mode, the SFC program runs automatically and the transitions are stepped through by the program. In manual mode the operator is able to control the processing of steps and transitions by using the carry out button.

In manual mode, there are three types of Inching available for preselection as follows:

- Actions and transitions are not activated
- Actions are activated
- Actions and Transitions are activated

So special care should be taken while creating the logic, if the user wants to run program in both automatic as well as manual mode.

See also ***Engineering Manual System Configuration, Commissioning***, ***Engineering Manual Operator Station Configuration*** and ***Engineering Manual User Access***.

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 through their own assignment editor, and these displays can be then called on the Freelance operator station

through the context menu of the step or transition. 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.

9.1.1 Create an SFC program

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

> Project tree > select a task > **Edit > Insert next level > SFC program**

9.1.2 Call SFC program editor

A program can be opened by selecting the SFC object in the project tree. This can be opened from the Edit menu or double-click the SFC program. The SFC program is opened as separate tab in the right pane. *T*his can be closed from the close button available at the right side of the opened tab.

> Select the SFC program from the project tree > **Edit > Program**

or

> Double- click the SFC program

The program is displayed with its current content (steps, transitions, etc.) and can be modified.

9.1.3 Close SFC program

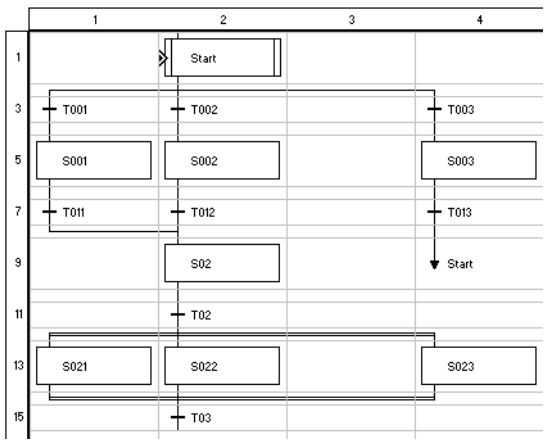
> Editor > Close

Closes the active SFC tab.

9.1.4 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.

9.1.5 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)**.

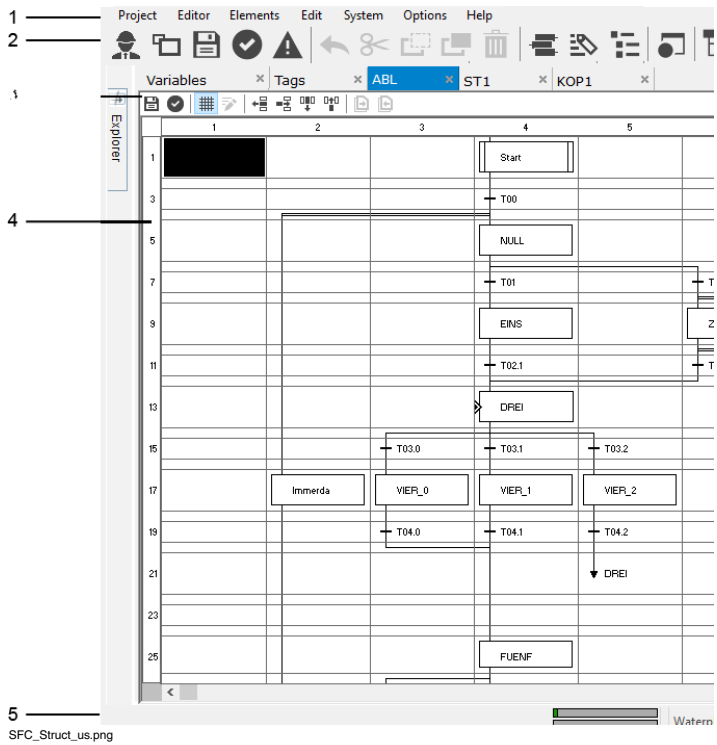
The other simultaneous steps are begun with **simultaneous sequence divergence add (> line 12, column 2+4)**.

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)**.

9.2 Structure of the Sequential Function Chart

9.2.1SFC program user interface



SFC_Struct_us.png

(1) Menu bar The menu entries are adapted to the active window or editor in Freelance Engineering.

(2) Common toolbar
accessible from the Project Explorer and the. Editor region.

(3) Editor toolbar
Frequently used commands of SFC are accessed while working in the SFC editor.

- Save editor
- Check editor
- Grid on/off

- Edit parameters of element
- Delete row
- Insert row
- Delete column
- Insert column
- Find next cross reference
- Find previous cross reference

(4) 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).

(5) Status bar The status bar indicates the name of the program which is being edited and name of the user.

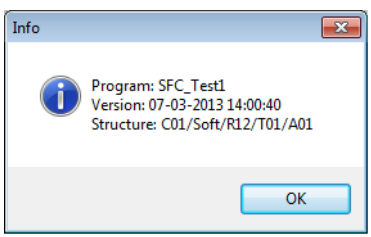
9.2.2 Display program information

Program version and position in the project structure

> Options > Version

The program name, date of last program modification as version identification and the structure path in the project tree are shown.

The structure path can be displayed in a **long** or **short format**, as set in the **Options** menu of the project tree.



di1430us.png

Program state

The status bar indicates the name of the currently edited program, the position in the project tree, the current user and license information.

9.2.3 Drawing help

Grid

> **Options** > **Grid**

With this menu item the grid of rows and columns can be displayed or hidden. A hook stands in front of the menu item if the grid is shown.

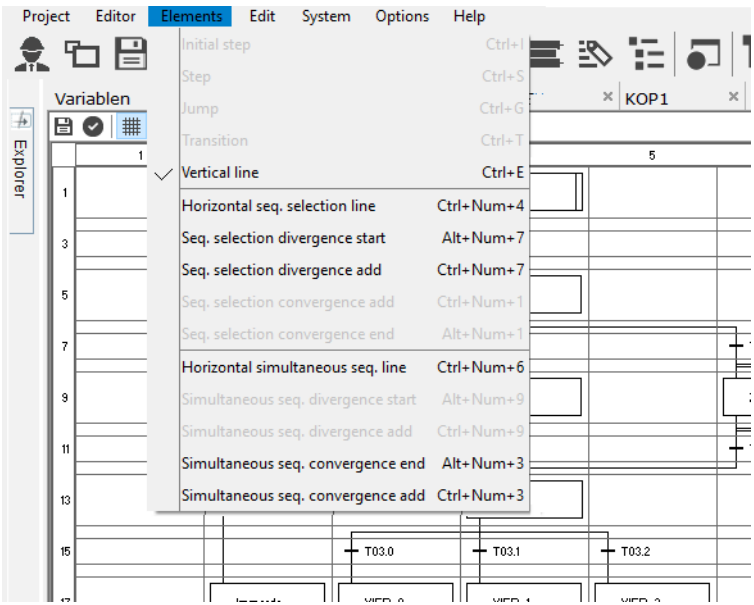
Row and column numbers

> **Options** > **Row and column numbers**

This menu item is used to show and hide the number of lines and columns. A hook stands in front of the menu item if the numbers are shown.

9.3 Edit SFC Elements

> **Elements**



SFC_Element_us.png

All SFC elements can be called from the **Elements** menu by right-clicking the editor from the shortcut menu, and also from the corresponding category in the Library explorer. Depending on the position in the editor, only the respective admissible elements can be inserted.

> **Elements** > select required SFC element

or

> Open category in Library pane > select required SFC element

or

> Right-click the editor > select required SFC element

In the following, only the selection from the menu is described.

Additionally the following rules are valid:

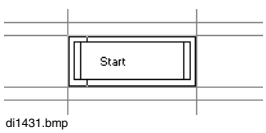
- Initial step, Step, Jump and Transition can only be entered into the thick lines.
- The hook in front of an element represents the default value and shows which elements can be entered by pressing the SPACE key. The default value is stored line by line, so that the last element entered in a line can be recalled with the SPACE key.
- The given names (max. 8 characters) must be unique within the SFC program.

9.3.1 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**

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



di1431.bmp

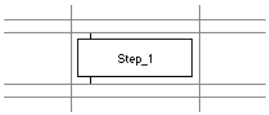
In the case of a sequence selection following directly after an initial step, it is not allowed to jump to the beginning of the sequence.

9.3.2 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 any of these programs the instruction list (IL), ladder diagram (LD), function block diagram (FBD), structured text (ST).

> **Elements > Step**

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.



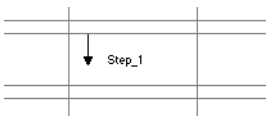
di1432.bmp

9.3.3 Jump

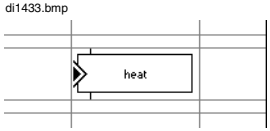
Jump is one means of going from one branch to another step situated inside or outside this branch. Jump is used instead of a step.Decisive for the execution of the jump is the transition at the jump target; this transition must be fulfilled.

> **Elements > Jump**

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.

9.3.4 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,

function block diagram (FBD), or structured text (ST). The result of the transition (.RESULT) must be logic 1, so that the following step(s) can be activated.

> **Elements > Transition**

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

di1434.bmp

9.3.5 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**

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.

9.3.6 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**

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

9.3.7 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**

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


di1439.bmp

Each started sequence selection divergence must be closed again.

9.3.8 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**

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

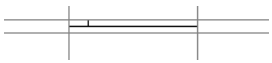

di1438.bmp

9.3.9 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

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



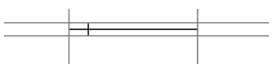
di1442.bmp

9.3.10 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

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

9.3.11 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

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.

9.3.12 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

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

Each started simultaneous sequence divergence must be closed again.

9.3.13 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

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

9.3.14 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

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

9.3.15 Simultaneous sequence convergence add

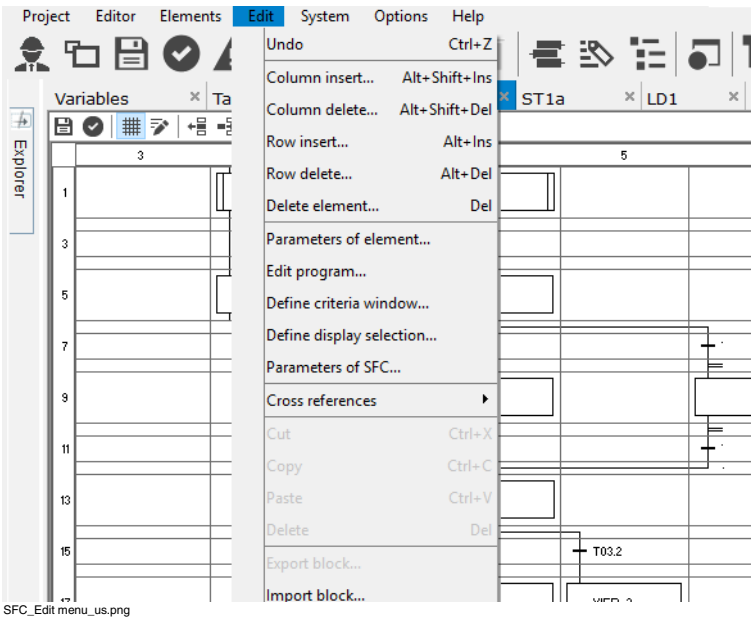
Apart from the end of a simultaneous sequence divergence, there is the element Horizontal simultaneous sequence line with which the parallel string is closed.

> Elements > Simultaneous seq. convergence add

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.

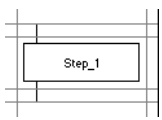
9.4 Edit SFC structure

> **Edit**



SFC_Edit menu_us.png

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.



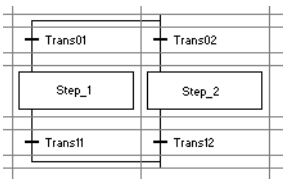
di1446.bmp
Element not selected

di1447.bmp
Element selected

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

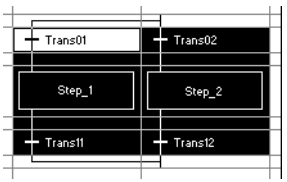
- Click the first element with the mouse. With a mouse key pressed only a frame can be drawn to contain all elements to be selected. If the frame is drawn only within the first element, this is marked as a block (edges of the field will be black).

- 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

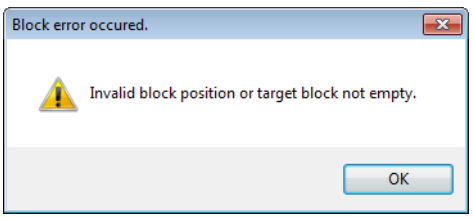


di1449.bmp

Block selected

9.4.1 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.



di1450us.png

9.4.2 Undo

> Edit > Undo

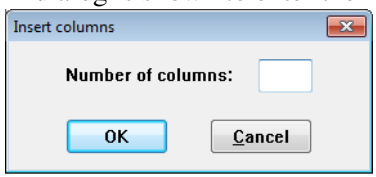
Only the last edited block function, i.e. **cut, delete** or **insert**, can be undone.

9.4.3 Edit columns / lines

Column insert

> **Edit > Column insert...**

A dialog is shown to enter the number of columns to be inserted.



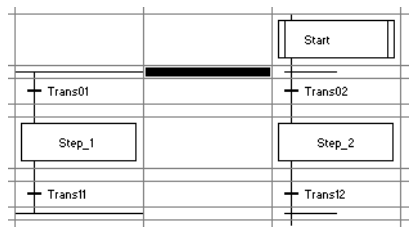
di1408us.png

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 process can be canceled. A message will also be given if, due to the insert, elements must be shifted right out of the grid (maximum of 16 columns).

di1451.bmp

After inserting a column



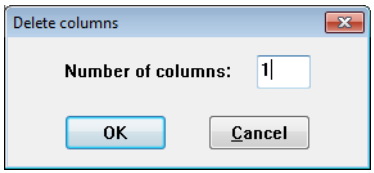
di1452.bmp

After inserting a column

Column delete

> **Edit > Column delete**

A dialog is shown to enter the number of columns to be deleted.



di1409us.png

The elements right of the selected column are shifted to the left by the given number of columns.

Only columns can be deleted which do not contain any elements.

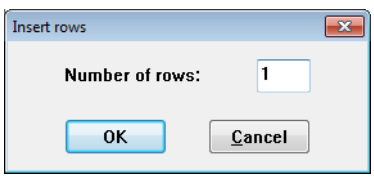
di1453.bmp
Before deleting a column

di1454.bmp
After deleting a column

Insert row

> Edit > Row insert

A dialog is shown to enter the number of rows to be inserted.

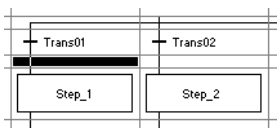


di1410us.png

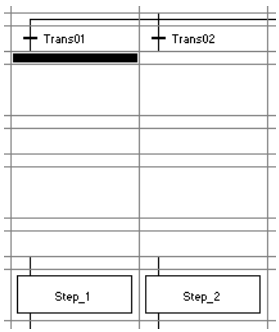
Insertion of lines can be done anywhere in the structure. The selected row is shifted downwards by the number of rows to be inserted. If the number of rows to be inserted is too big, an acoustic signal will follow and the insert will be rejected. A smaller number can now be stated or the process can be canceled. A message will

also be given when due to the insert elements must be shifted down out of the grid (maximum of 512 lines).

The general structure of an SFC program cannot be changed; in any case four numbered rows - a step, a connection line, a transition and another connection line - form a unit.
Thus for inserting rows, always four numbered rows are counted as one row to be inserted.



di1455.bmp
Before inserting two rows

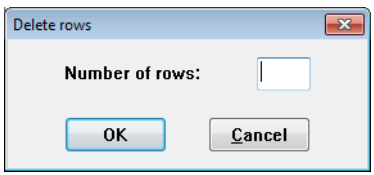


di1456.bmp
After inserting two rows

Delete row

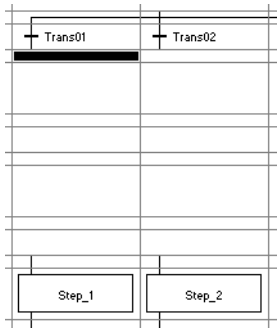
> **Edit > Row delete**

A dialog is shown to enter the number of rows to be deleted.



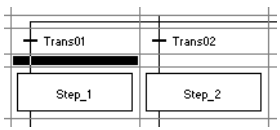
di1411us.png

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

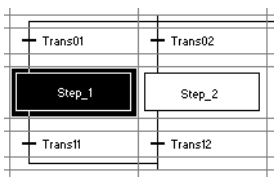
After deleting two rows

Only empty rows can be deleted; a block of four empty lines must be empty to delete one row.

Delete element

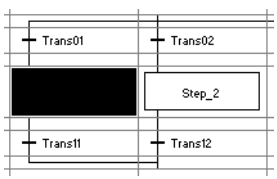
> **Edit > Delete element**

Each element or field of the SFC structure can be deleted. The selected element is deleted after a confirmation by the user.



di1460.bmp

Before deleting an element



di1461.bmp

After deleting an element

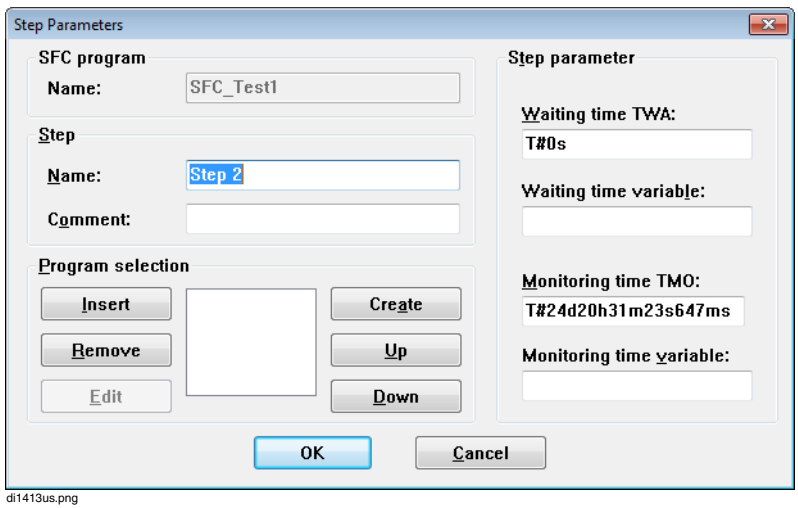
9.4.4 Parameterize SFC program elements

Parameterize steps

> **Edit > Parameters of element**

> Double-click on step

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



di1413us.png

SFC program

Name Displays the name of the SFC program

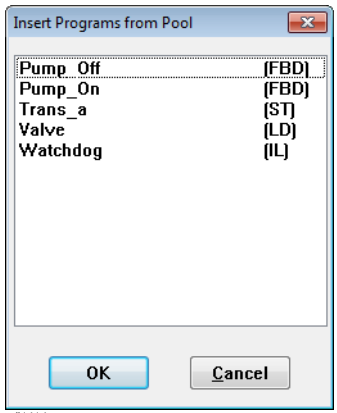
Step

Name The step name can be edited.

Comment The short comment of the step can be edited.

Program selection

Insert



di1414us.png

All FBD, LD, IL, or ST programs that are available in the project pool, can be selected. They will be processed 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 program list and stored in the pool.

Edit The program selected in the step program list is called up in the corresponding editor (IL, FBD, LD, ST), so that modifications can be done. Before the editor is called up, however, the SFC program must be stored.

Create A new program can be created. To do this, the desired type of program is selected in the **object selection** dialog 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 FBD, IL, LD or ST editor.

Up The program selected is moved up for one position in the processing sequence.

Down The program selected is moved down for one position in the processing sequence.

Step parameters

Waiting time TWA:

The **waiting time TWA** is the minimum dwell of the SFC program in one step. Even if the transition is fulfilled subsequent transitions will be enabled only after expiry of the TWA. A constant value as well as a variable may be entered. If both values are entered the current value of the variable will be taken during run time.

Monitoring time TMO:

The **monitoring time TMO** is the maximum desired dwell in a step. On overshooting the TMO an alarm message with the configured priority level 1 ... 5 (see Parameterize SFC program on page 330) will be generated. A constant value as well as a variable may be entered. If both values are entered during run time the current value of the variable will be taken.

If the step parameters TWA or TMO are changed directly in the parameter definition dialog of the step and loaded during processing of the SFC program, this leads to an initialization and restart of the SFC program.

Use the waiting / monitoring time variables instead of changing the step parameters via the configuration to avoid restarting the SFC program.

Structured variables cannot be used together with step parameters of elements in the SFC. Please use standard data types instead.

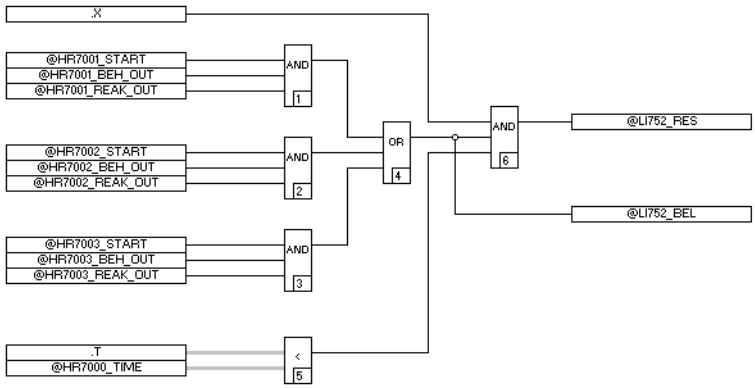
Step program

In each step program the two local SFC variables **.X** and **.T** may be used. These variables are only valid within a particular step and have the same names in all steps.

The variable **.X** shows the current state of the step. If the step is active the variable**.X** is set to TRUE. When the subsequent transition is fulfilled the programs of the step are calculated once again. The variable **.X** is then set to FALSE. If actions are to be reset, this can be done with this variable.

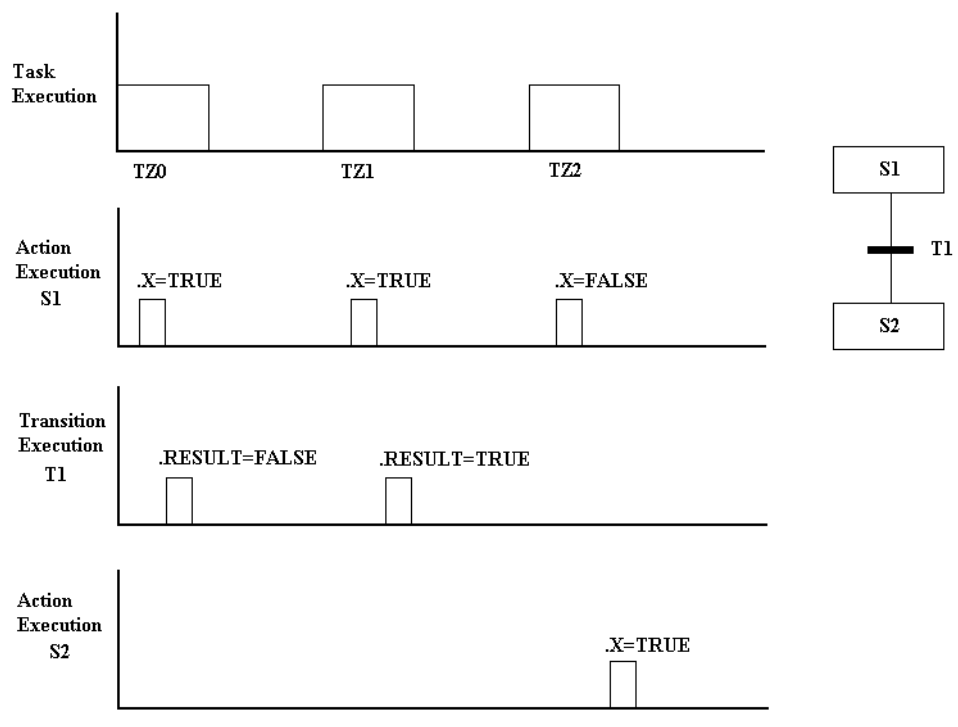
The current step run time is available in the local SFC variable **.T**.

Example of a step program



di1462us.bmp

The following example shows how the steps are calculated after the transition is switched.



tp002us.bmp

With Transition 1 fulfilled 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.

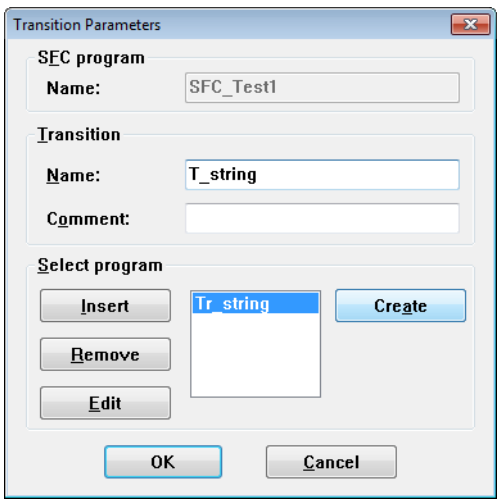
Parametrize transitions

> **Edit > Parameters of element**

or

> Double-click transition

If a transition has been previously selected, the parameter dialog for transitions is displayed:



di1416us.png

SFC program

Name: Displays the name of the SFC program

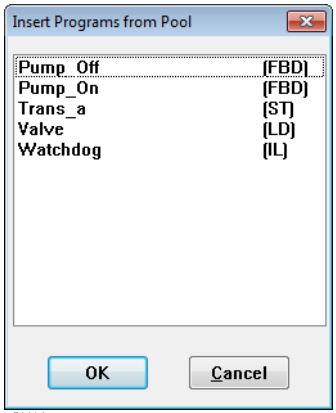
Transition

Name: Transition name can be edited

Comment: Short comment of the transition can be edited

Select program

Insert



di1414us.png

Only **one** of the FBD, IL, LD or ST programs available in the pool can be selected. It will be processed when the selected transition is active. The desired program is selected and inserted in the transition program list with the OK button.

Remove The selected program is removed from the transition program list and stored in the pool.

Edit The program selected in the transition program list is called up in the corresponding editor (IL, FBD, LD, ST), so that modifications can be done. Before the editor is called up, however, the SFC program must be stored

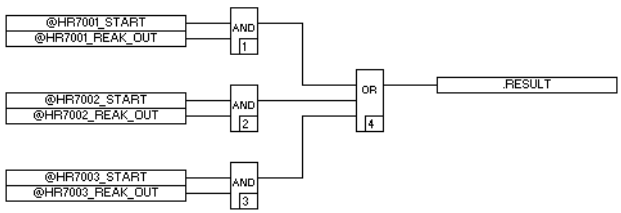
Create A new program can be created, when no other program has been inserted into the **program list** of the transition. To do this, the desired program is selected in the **object selection** dialog and added to the **program list** of the transition with OK. To be able to continue editing in the appropriate editor see **> Edit**.

Transition program

In each transition program the local SFC variables **.RESULT** must be set. This variable is only valid within a particular transition and has the same name in all transitions.

A **transition** is successfully concluded, when the variable **.RESULT** is set to TRUE. If the variable **.RESULT** is already set to TRUE during first calculation, the corresponding step is calculated once with **.X=TRUE** and in the next cycle with **.X=FALSE.**

Example of a transition program



di1463us.bmp

When the calculation of the translation is started, the variable **.RESULT** is initialized with FALSE. If **.RESULT** is used in a conditional command it must be ensured that **.RESULT** can be set to TRUE. In the following example **.RESULT** is not set to TRUE and, therefore, the transition is never fulfilled:

```
IF i<5 AND i>9 THEN
  .RESULT := TRUE;
END_IF;
```

9.4.5 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 (FBD, LD, IL, or ST).

9.4.6 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 the Freelance operator station at the appropriate step can be selected. Variables can be entered on the one hand from the system wide variables list, on the other hand, from the programs assigned to the step. To this end, however, the corresponding programs (FBD, LD, IL, or ST) 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 =** Yes must have been configured for this purpose.

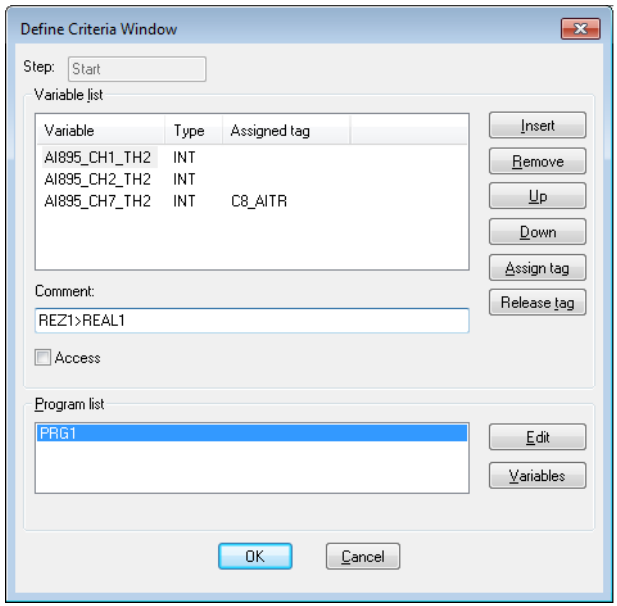
A tag can also be assigned to each variable. This enables a relevant faceplate to be called up on the operator station in the operation dialog of the criteria window.

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.

Define step criteria window

> Edit > Define criteria window

If a step has already been selected, the dialog for setting parameters for the step criteria window is displayed.



di1468us.png

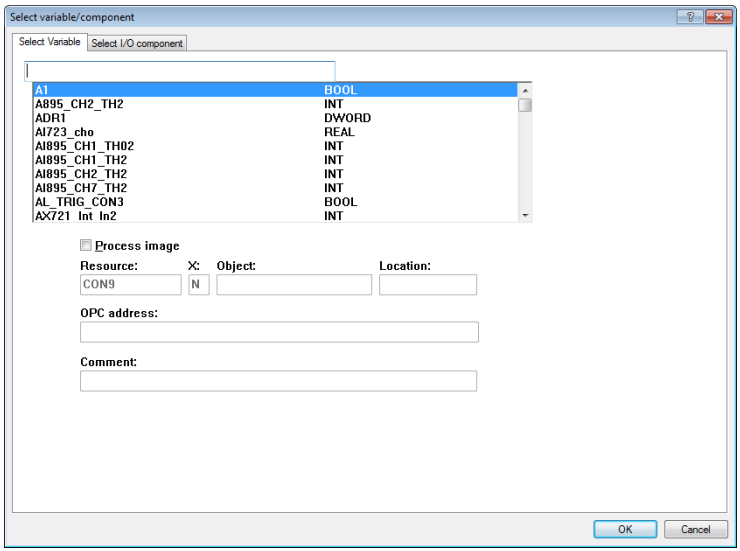
Step Name of the selected step, cannot be changed here

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



di1469us.png

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 (Freelance Operations). The tag's faceplate then makes direct operator action on that tag possible.

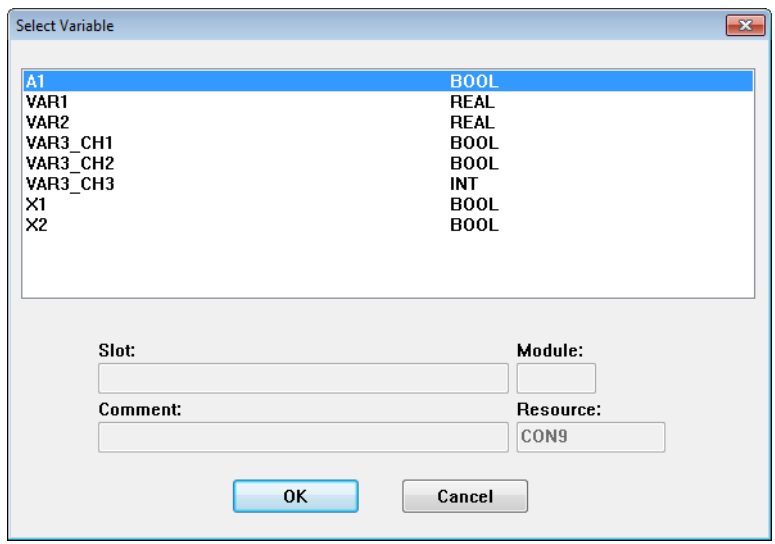
Release tag The tag assigned to the variable is released again. It is thus not possible in the user interface to select any tag's faceplate for this variable.

Program list

Edit

The program selected (highlighted) in the **program list** is called up in the appropriate editor (FBD, LD, IL, or ST) 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



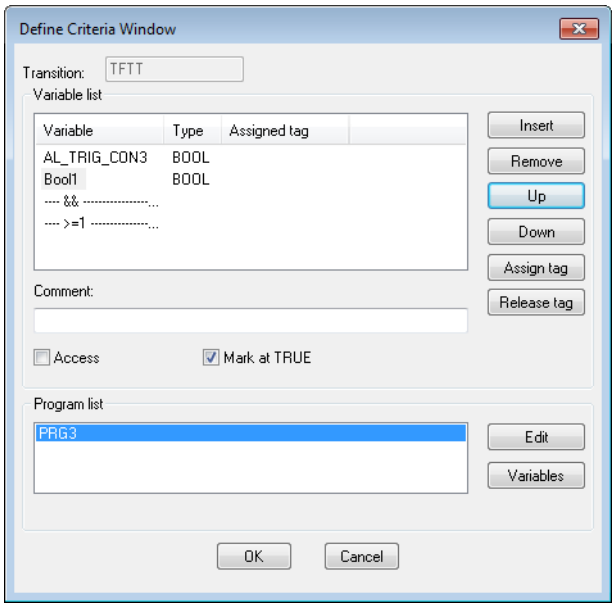
di1470us.png

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.

Define transition criteria window

> Edit > Define criteria window

If a transition has already been selected, the dialog for setting parameters for the transition criteria window is displayed.

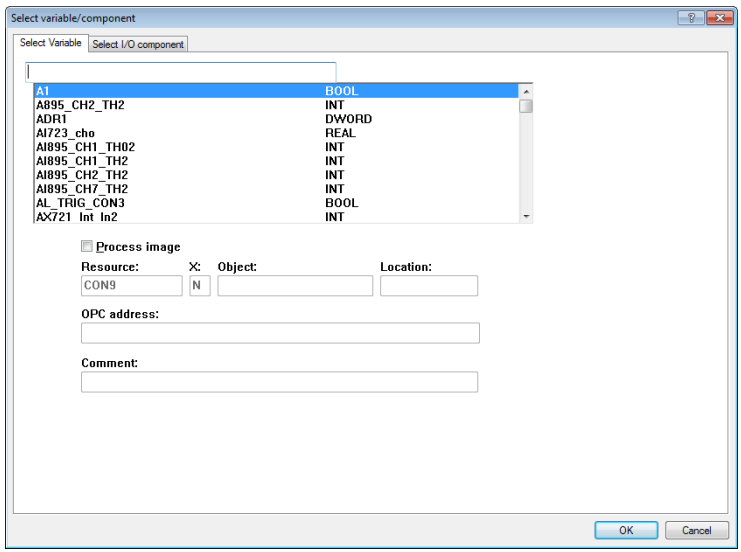


di1471us.png

Transition Name of the selected transition, cannot be changed here

Variables list

Insert



di1469us.png

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 (Freelance Operations). 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.

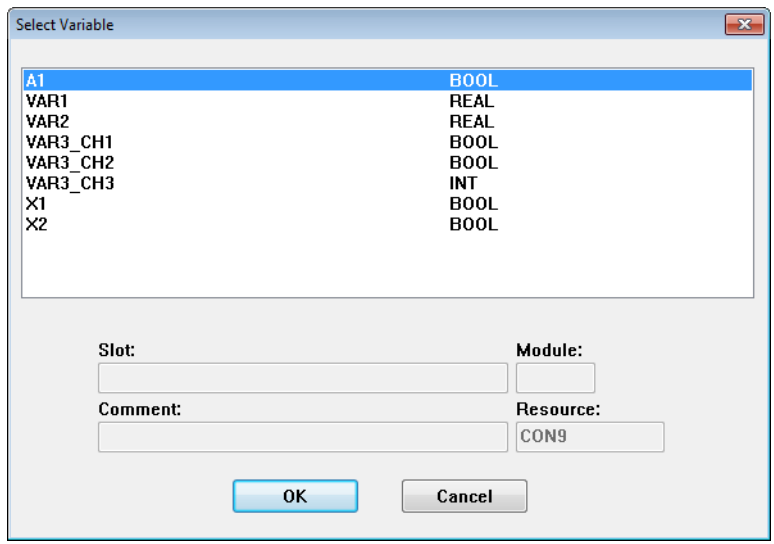
Access ☑ Write (operator intervention) of the selected value is permitted.

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 FBD) 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



di1470us.png

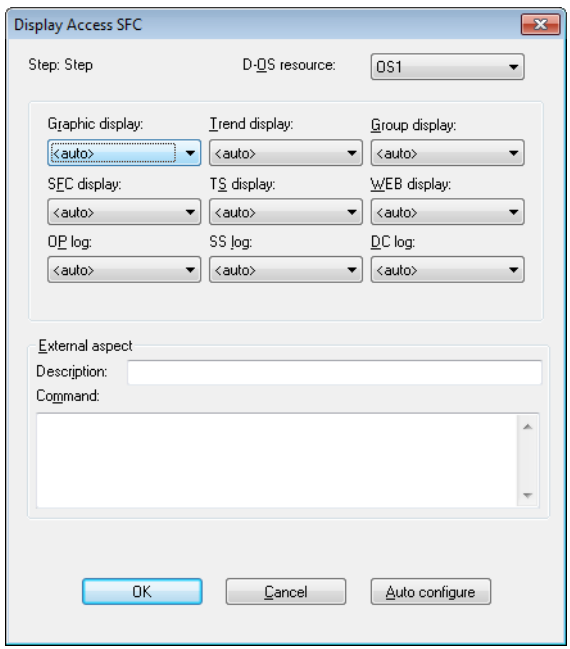
A list is displayed containing all the variables belonging to the program selected under the step's **program list**. Only **one** variable may be selected. Comment and resource name are also displayed.

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.

9.4.7 Define display access

Displays and logs might be allocated for each transition and each step. Thus, it is possible for the operator to call these via the context dialog on the operator station. Here, either all displays or only those of one operator station are presented.

> Edit > Define display selection



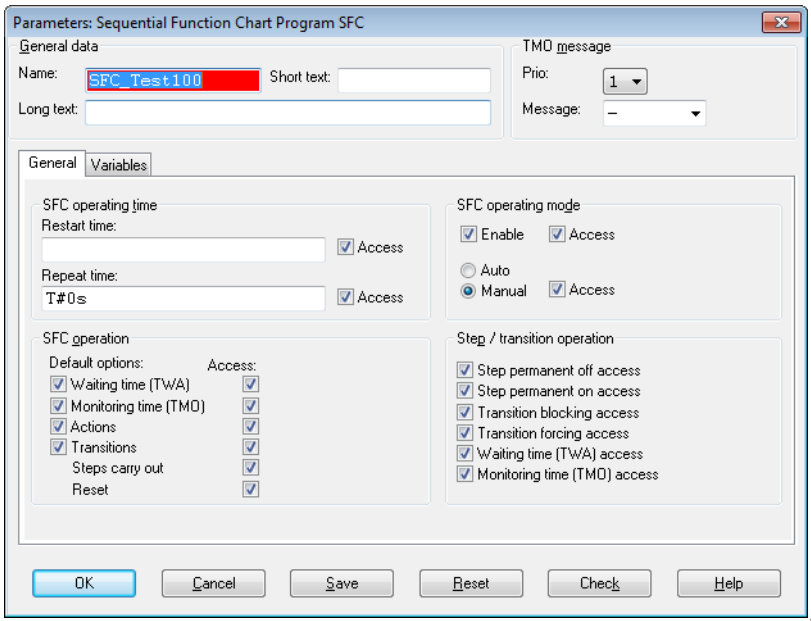
di1473us.png

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 Configuration, Standard displays***.

9.4.8 Parameterize SFC program

> Edit > Parameters of SFC



di1421us.png

General data

Name Name of the SFC program; the name will be included in the tag list.

Short text Short text to the SFC program; up to 12 characters, all characters are permitted.

Long text Long text to the SFC program; up to 30 characters, all characters are permitted.

TMO message

Prio Priority level 1 to 5 for message “Monitoring time overflow”.

Message Select message text from list or enter free text.

Upon overshooting the monitoring time TMO, an alarm message with the selected priority level and the configured 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

Activation of daylight saving time affects the display of time points. When operator enters a time value, the user must specify whether the edited time is a daylight saving time or not. A daylight saving time must be identified by an “S” as suffix following the time value. If this “S” is missing, the time value input is interpreted as local time. If an “S” is specified for a time when daylight time is not in effect, a message is sent to the user requesting a correction. For example, Input “..16:00..” produces 16:00 at the station; an input of “..16:00..S” produces (daylight saving time)15:00 for CET time zone.

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 against the repetition time. If the repetition time is smaller or equal to the runtime of the sequence control, the sequence control is restarted 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 against 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 Freelance Operations.
Configuration of variables of type BOOL allows the program to take control over switching manual and automatic mode.

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 dialog. 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 Parameterize SFC program on page 330) 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

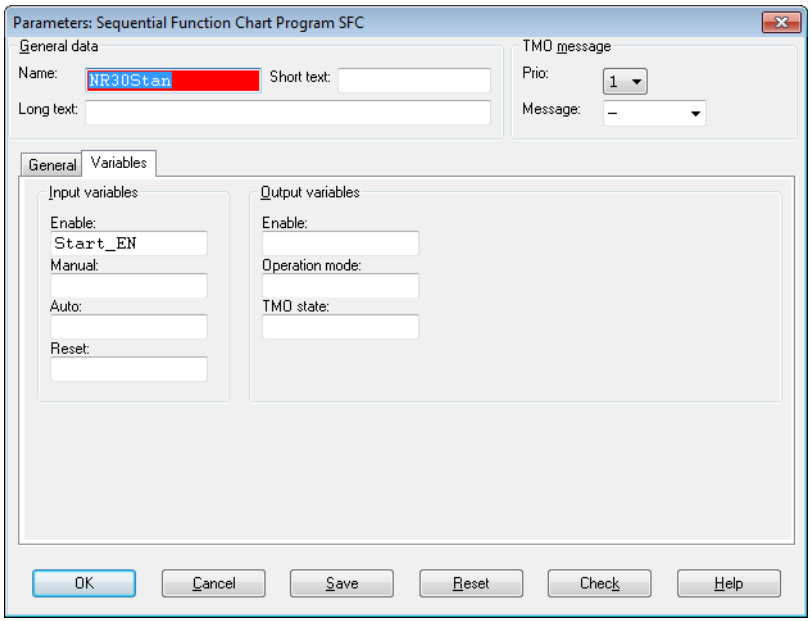
The waiting time in a step can be changed by the operator at the operator station.

Monitoring time (TMO) access

The monitoring time can be changed by the operator at the operator station.

Variables of an SFC program

> Edit > Parameters of SFC >



di1472us.png

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-a-vis the modes operation parameter, the Manual variable is dominant vis-a-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 according 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.

9.4.9 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

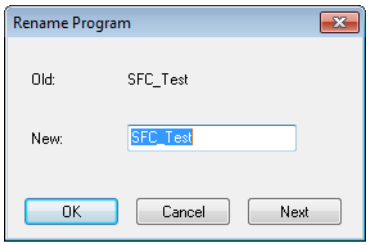
The selected elements are stored in a buffer memory and can now be inserted at another place.

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.



di1424us.png

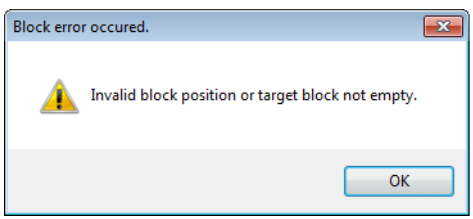
New Insert new name

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:



di1425us.png

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.

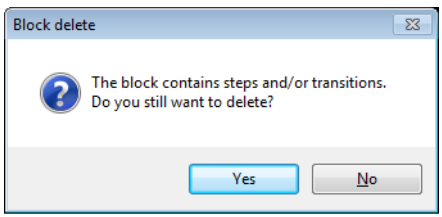
After cutting the parameterized elements, their parameters are reset after the insert, i.e. inserted programs become available again in the POOL.

If the parameter definition should be retained, do not shift it with Cut - Paste (see Shift blocks on page 310).

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.



di1422us.png

9.4.10 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 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”.

Import block

> **Edit > Import block**

This is used to import the contents of a block from a **.as-file** which had previously been generated with export block to the preselected position in the SFC program.

9.4.11 Program administration functions

Save the program

> Project > Save Tab

The program is saved without exiting. Programs that are not correct can also be saved and then completed at any time.

If the project is not saved in the project tree on closing or before, changes made to the program are ineffective.

Document the program

> Project > Documentation

The editor for documentation is opened as a separate tab in the right pane. It can be closed using the Close button available at the right side of the opened tab, and reopened later from the main menu.

Documentation administration is opened. This is where user specific project documentation is defined and output. For a description, see ***Engineering Manual System Configuration, Documentation***.

Program header

> Project > Header

A program-specific short comment can be added to the program documentation header, or this can be edited.

For drawing header / footer, see ***Engineering Manual System Configuration, Documentation***.

Edit program comment

> **Project** > **Comment**

A longer program-specific comment can be edited here to describe the functionality. For a description, see ***Engineering Manual System Configuration, Project manager***.

Print

> **Options** > **Print**

The contents of the screen are output to the standard printer.

Plausibility check

> **Editor** > **Check**

All inputs relevant to operation are checked for syntactical and contextual correctness. Errors, warnings and notes that are found are displayed in an error list. If the plausibility check detects errors, the processing state of the program is **implausible**.

The processing state of program elements that are newly entered, copied or moved is **implausible**.

This plausibility check reviews the accuracy and consistency of the program itself. To test the correctness in the project context call plausibility from the project tree. See Engineering Manual System Configuration, Project Tree, plausibility.

Error list

> Editor > Show error list

Any errors present in the program is displayed in the error list. Double-click a check message to jump to the line in the program that caused this error.

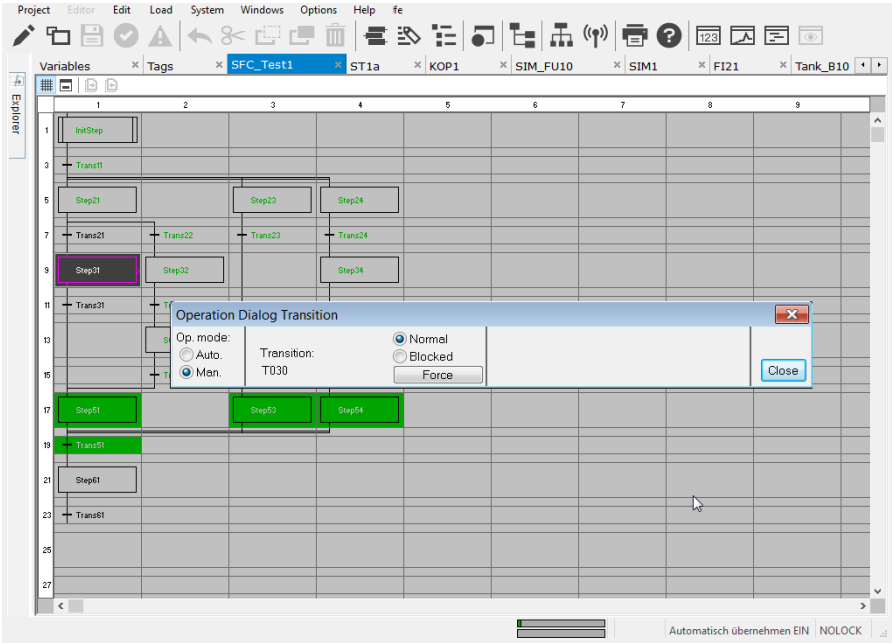
See also ***Engineering Manual System Configuration, Project tree***.

9.5 Commissioning the 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 Freelance operator station, 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.



di1581us.png

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.

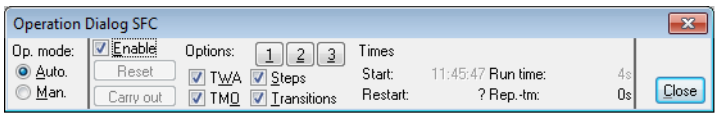
In contrast to Freelance operator station, it is also possible to change the state of steps and transitions in automatic mode.

In commissioning, it is not possible to change the structure of the SFC program. This is only possible in configuration.

.Each new load of the SFC program results in a restart with the initial step.

9.5.1 Operation dialog SFC program

> **Operation** > **SFC program**...



di1552us.png

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, are stepped through once. The processing of steps or transitions is controlled by the Options setting (check boxes of TWA, TMO, steps and transitions).

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

1 TWA, TMO, actions and transitions are not activated.

2 Actions activated.

3 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.

Activation of daylight saving time effects the display of time points. When operator enters a time value, the user must specify whether the edited time is a daylight saving time or not. A daylight saving time must be identified by an “S” as suffix following the time value. If this “S” is missing, the time value input is interpreted as local time. If an “S” is specified for a time when daylight time is not in effect, a message is sent to the user requesting a correction. Example: Input “..16:00..” produces 16:00 at the station; an input of “..16:00..S” produces (daylight saving time)15:00 for CET time zone.

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

Changes are made by entering the new value from the keyboard. The entry must be in IEC 61131 - 3 date and time notation.

The restart time is entered in the Date and Time format (DT):

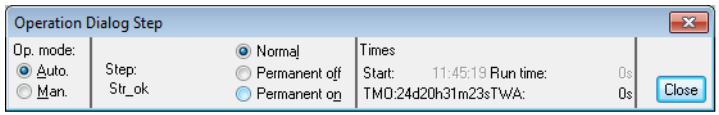
Example: DT#1999-12-31-23:59:59.99

The repeat time is entered in time format (TIME):

Example: T#3m30s

9.5.2 Step operating dialog

Select step with left mouse button > **Operation** > Step...



di1358us.png

Close The operation dialog step is closed.

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.

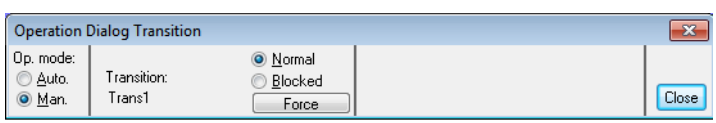
Run time The **run time** shows the time that the active step has been active. With each new execution of a step, the run time is reset to 0s

TMO ***Monitoring time*** for this step. If this time is exceeded, a message is generated.

TWA **Minimum waiting time** for one step.

9.5.3 Transition operation dialog

> Select transition with left mouse button > **Operation** > **Transition**...



di1359us.png

Close The operation dialog for transitions is closed.

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. The processing of the transition is controlled by the Options setting of the SFC (refer to Operation dialog SFC program). Forcing is also possible for a blocked transition.

9.5.4 Step states

Steps in the Freelance system can have the states **inactive, active, was active and faulty**.

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.

9.5.5 Step action execution

The Freelance system offers the user the modes **normal, permanent off** and **permanent on** for the execution of step-related actions.

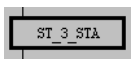
Action execution is independent of step state.

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.

9.5.6 Display of steps in the SFC program

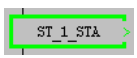
The appearance of steps in the SFC program display depends upon the step state and the action execution mode.

initial step



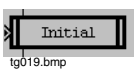
normal step

permanent off step



permanent on step

jump out



jump in

Colors used to display steps depending on their state and action execution mode.

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

9.5.7 Transition states

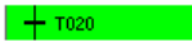
In the Freelance system, transitions can take on the states **not enabled, enabled, fulfilled** or **completed**.

not enabled	Not all preceding steps have been active > transition criteria not evaluated
enabled	All preceding steps have been active > transition criteria evaluated
fulfilled	The transition criteria are fulfilled. All preceding steps become inactive and all dependent steps become active - transition is made.
completed	All dependent steps are active, transition has been completed.

9.5.8 Display of transitions in the SFC program

The appearance of transitions in the SFC program display depends on their state.

enabled or **fulfilled**



not enabled or **completed**

Colors used to display transitions depending on their state.

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	dark blue	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	black	dark green
completed	Background	gray	gray	gray
	Lines	black	dark blue	black
	Text	dark green	dark blue	dark green

10 User Function Blocks

10.1 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 defined 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 dialog.

The configuration of a user function block class is done in the project tree under **Function block pool P-FB**. Every user function block class gets a class name which can be freely chosen; with this name the block can be called from inside other programs.

The user function block program can be written in one of the programming languages Function Block Diagram (FBD), Ladder Diagram (LD), Instruction List (IL) or Structured Text (ST). 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, a user function block can be used. It can then be found for later use under the **Elements > Blocks > User function blocks** menu.

For actual use, **instances** of a user function block class must be created. Each instance has a parameter dialog, which contains at least a tag name, short text and long text. Each user function block instance must be assigned a unique tag name.

User function blocks can be called from all programming languages: IL, LD, FBD, ST and SFC.

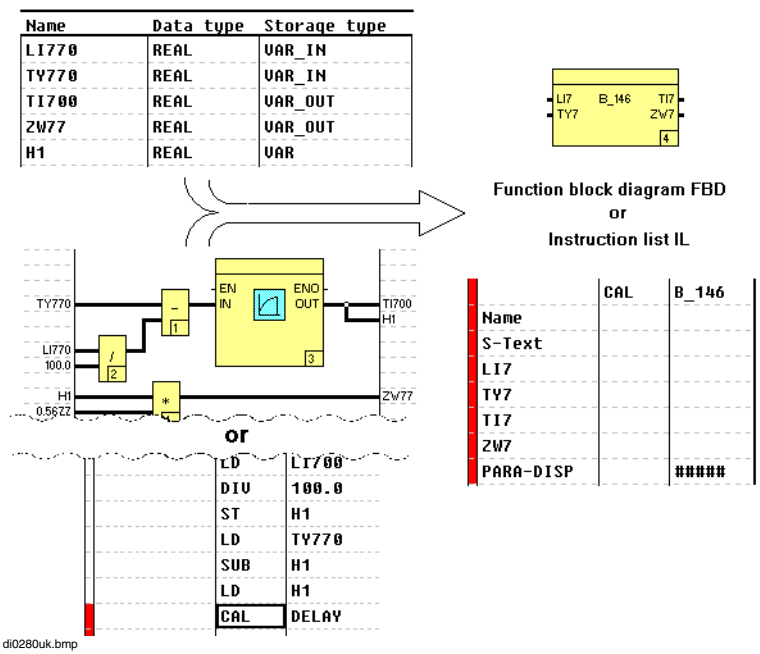
Changes to user function blocks are made to the function block class and effect all function block instances of that class in place. If new pins are added to a block, the function block instances must be replaced and the new pins must be connected.

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.

For licensing of user function blocks refer to ***Getting Started Manual***.

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



Creation of a user-defined function block:

- In the 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, FBD or ST
- 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*

10.1.1 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.

10.1.2 Create user function block pool

> Select SOFTWARE node from project tree

> **Edit** > **Insert next level**

> Choose **User Function Block Pool P-FB** from object selection window

> Enter pool name, max. 4 characters

Only one user **function block pool** may be created per project. All user function blocks are present in this; an unlimited number of user function block classes can be declared.

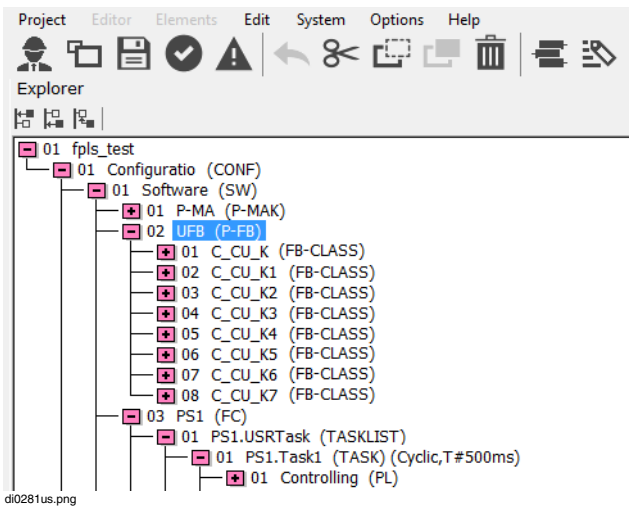
10.1.3 Create a user function block class

> Select User function block pool (P-FB) from project tree

> **Edit** > **Insert next level**

> Select **User function block class** from object selection window

> Enter class name



di0281us.png

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 naming conventions and must be unique on a project-wide basis. As UFB class names are used in the editors for programming the application, no special characters such as

+ - * / & = < > [ ] . , ( ) : ; ’ @ # $

may be used in the names.

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.

10.1.4 Create a user function block program

> Select class name FB-CLASS from project tree

> **Edit** > **Insert next level**

> Choose either **FBD, IL, LD** or **ST program** from object selection window

> Enter program name

The program to be executed as a user function block consists of one and only one FBD, LD, ST or IL program. A combination of programs of different types is possible by nesting user function blocks inside one another.

For details how to define the program of a user function block refer to User function block program on page 374.

10.1.5 Create a user function block faceplate

> Select class name FB-CLASS from project tree

> **Edit** > **Insert next level**

> Select **Faceplate FB-FPL** from object selection window

> Enter faceplate name

Instance-specific values of a user function block instance can be displayed in Freelance operator station 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.

10.2 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

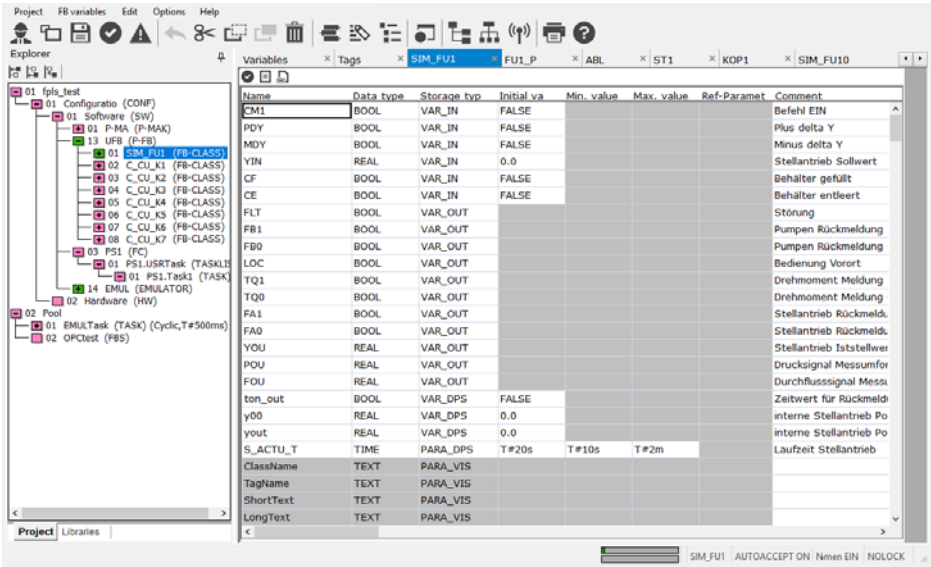
10.2.1 Interface of a user function block

Interface editor

The variables used in the user function block program must be entered in the **Interface editor** under **User FB variables.** The parameter dialog is created and the text list administered in the **Interface editor.**

> In User FB program > **System> User FB variables**

> Double-click User FB class node (FB CLASS)



di0283us.png

Legend:

:= May be edited

:= May not be edited and pre-allocated or no reasonable entry possible

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 choose 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.

Names of the user FB variables

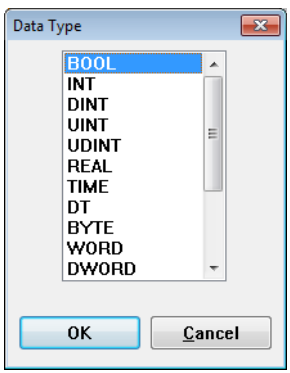
For values with storage types **VAR_IN** and **VAR_OUT,** the first three characters of the name become the pin identifier. Upper or lower case are both allowed and are differentiated. The first three characters of two pins may not be identical, otherwise a plausibility warning results.

Instances of user function blocks with identical pin names may only be used to a limited extent in other programs.

The order in the declaration follows the displayed pin layout in the user function block instance.

All defined variable names are valid only within the user function block class in which they were defined.

Data type



th001us.png

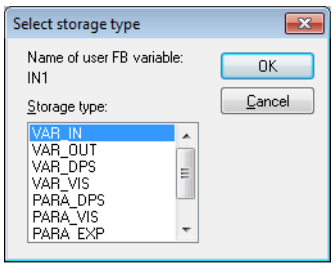
All Freelance data types are allowable as user function block variables. User-defined data types cannot be used.

The additional data type UNICODE-Text is available for storage type PARA_VIS, only. 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



Select storage type_UFB_us.png

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. The values are calculated by the user function block program and can 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 can be commissioned i.e. they can be written 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)	(7)	Use
VAR_IN	PS	x	x	x	-	-	-	-	Input pin
VAR_OUT	PS	x	x	x	x	-	-	-	Output pin
VAR_DPS	PS	x	x	x	x	-	-	-	Internal variable on the process station
VAR_VIS	Op.	x	x	-	-	-	-	-	Internal variable of Freelance Operations
PARA_DPS	PS	x	x	x	x	x	x	x	Parameter on the process station

Storage type	Data source	(1)	(2)	(3)	(4)	(5)	(6)	(7)	Use
PARA_VIS	Op.	x	-	-	-	-	-	x	Parameter in Freelance Operations
PARA_EXP	PS	x	x	-	-	x	x	x	Nested FB parameter
MP_EXP	PS	x	-	-	-	-	-	x	Nested FB message data

Legend
PS Process station
Op. Freelance Operations
x Function available
- Function not available

(1) Read from Freelance Operations — Variable can be accessed from Freelance Operations.

(2) Write from Freelance Operations — Variable can be changed (WRITE) from Freelance Operations or through a gateway.

(3) Download to PS — This variable is loaded into the process station and may be used in the program.

(4) Write from PS — Variable can be changed by the program on the process station.

(5) Write from Freelance Engineering — Variable can be changed (WRITE) from Freelance Engineering in commissioning mode.

(6) Correct from Freelance Engineering — Variable can be “corrected” from Freelance Engineering in commissioning mode.

(7) Conf. in parameter dialog — The variable may be altered by Freelance Engineering in configuration mode (parameter dialog).

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. Each 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

10.2.2 Edit interface of a user-defined function block

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 active.

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 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 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 (user function block variables) are checked for plausibility.

Close

> **FB variables** > **Close**

This closes the editor.

Print

> **Options** > **Print**

The screen contents are output to a printer.

Color setting

> **Options** > **Color setting**

The color of the fields that cannot be edited by the user can be set

Save column settings

> **Options** > Save column settings

The column width setting is saved.

10.2.3 Parameter dialog of a user function block

Dialog editor

> **Edit** > Dialog editor

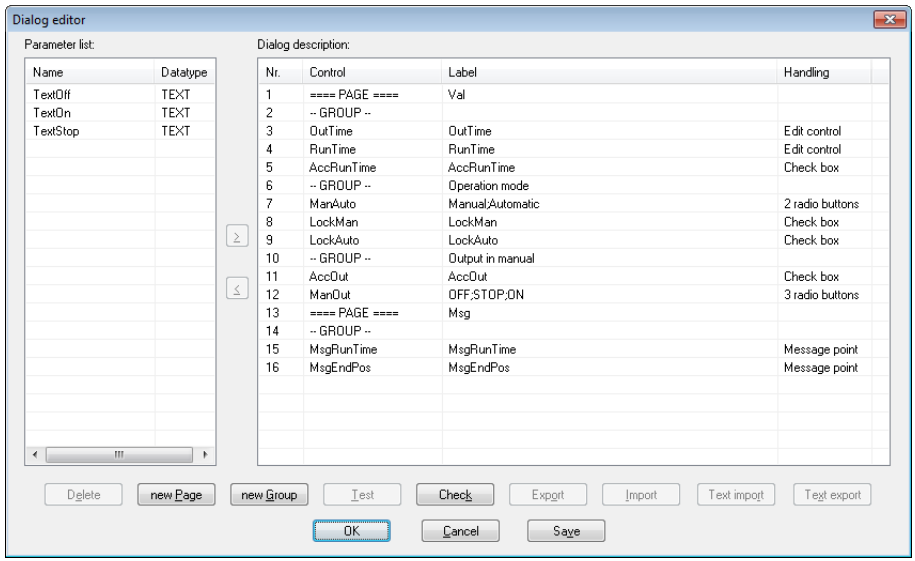
Every user function block class has a default parameter dialog. With this default parameter dialog the tag name and the short and long text can be configured.

The dialog editor can be used to create a customized parameter dialog for a user function block class. This parameter dialog can then be used to assign other instance-specific parameter values.

The elements for use in the parameter dialog 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 dialogs subdivided into group areas.

If the default dialog is extended, it must be given a new page.



th003us.png

The **Export**, **Import**, **Text Export** and **Text Import** buttons are not supported.

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	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 dialog by dividing the parameters into groups.
6. Run plausibility check
7. Test the parameter dialog

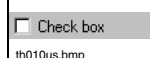
Variables that are to participate in the parameter dialog must be copied from the parameter list to the parameter description.

> 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
Input field	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. The entered data can be scrolled in the field. 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 point

The message point control 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:

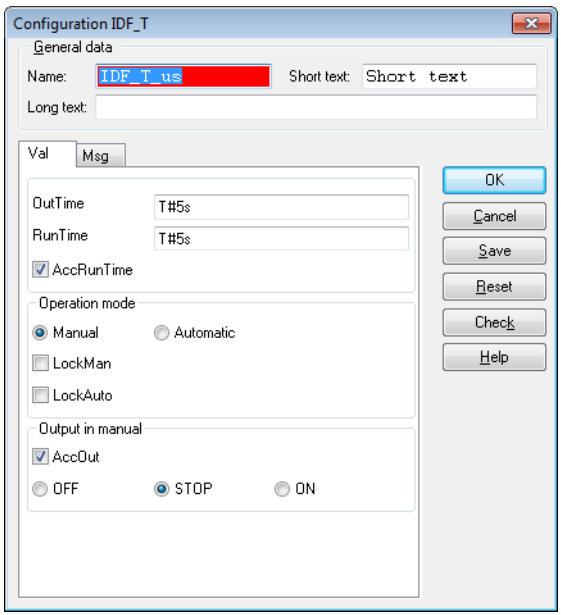
OK	Close parameter dialog and save values.
Save	Save values from the dialog.
Cancel	Close parameter dialog without saving.
Reset	Restore the saved values to the parameter dialog.
Check	Check the dialog for plausibility.
Help	Call up help text for the parameter dialog.

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



UserFB Parameter_us.png

10.2.4 Text list

Texts for display are required both in the configuration of a user function block instance in Freelance Engineering and in the graphic displays for operation and logging in the Freelance operator station.

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 Freelance Engineering, 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) in Unicode 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
```

Import text list

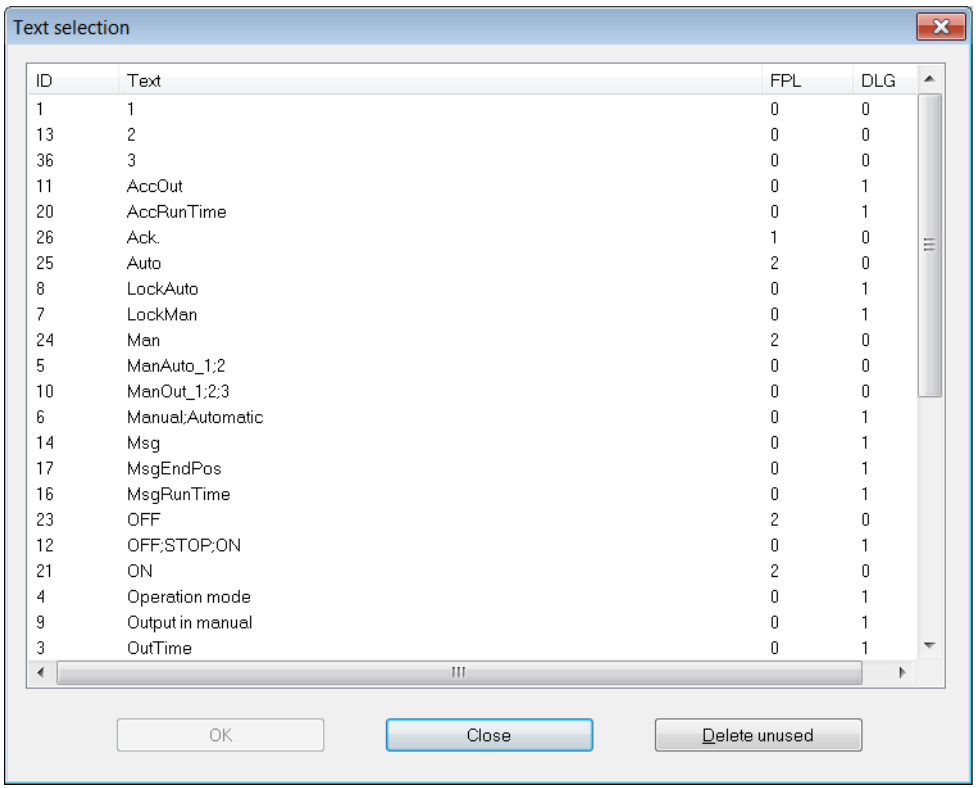
> **Edit** > Import text list

A file that has been previously saved can also be read in from a data carrier (e.g. hard disk). 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.png

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 Freelance Engineering.

10.2.5 User function block program

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, the drawing of connecting lines, shifting and plausibility checking are not different from configuration in user programs. For program creation, see for more information:

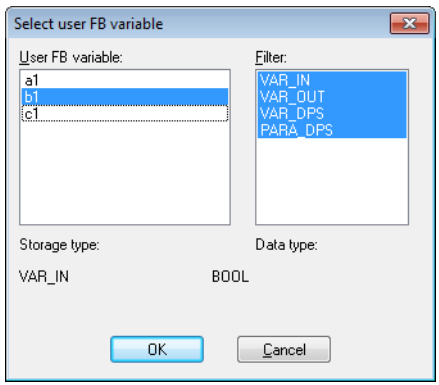
- Section 5, Function Block Diagram (FBD),
- Section 6, Instruction List (IL),
- Section 7, Ladder Diagram (LD), or
- Section 8, Structured Text (ST).

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



di0287us.png

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 the Freelance operator station (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.

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), refer to ***Engineering 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.

10.2.6 User function block faceplate

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).

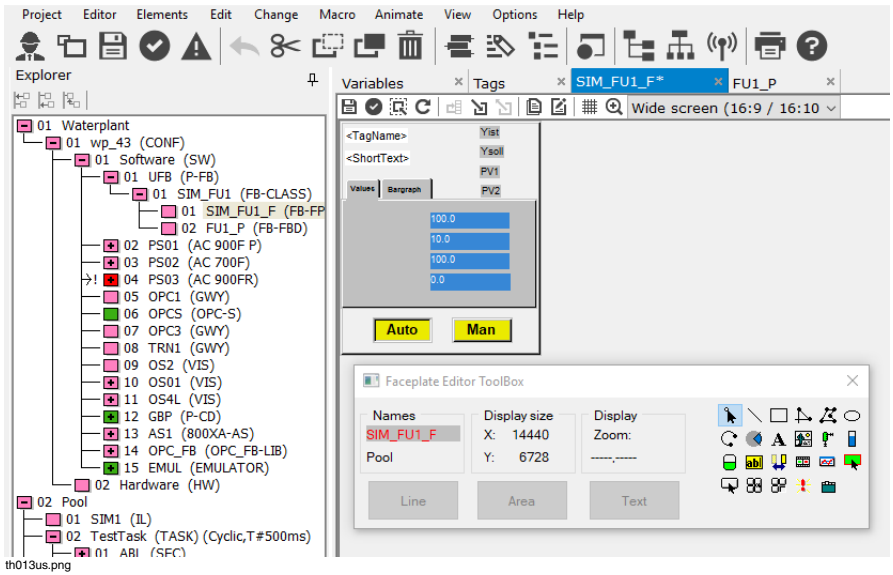
> Project tree > double-click on user function block faceplate

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 Configuration, Graphic display***.

Default static images are available for the overview faceplate. These images cannot be edited.



th013us.png

In principle, there is no difference between the graphic creation of a faceplate and that of a graphic display. The complete interface menus, dialog dialogs, hint texts, error messages is virtually the same as that of the graphic editor. (See Extensions in the faceplate editor on page 378).

It is not 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.

Extensions in the faceplate editor

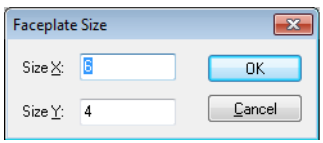
Faceplate size

A faceplate may have any rectangular size fitting within the 30 x 4 display format (**X-Size** up to30 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.png

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 Freelance Operations (storage type: PARA_VIS; data type: TEXT) can be displayed. New static texts will be added automatically to the text list.

> **Draw** > **Text** > **F2** key

th043us.png

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 on the Freelance operator station.

The content of a text parameter in the faceplate can only be changed by a load operation in commissioning mode in Freelance Engineering.

10.2.7 Check 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 Help for user function blocks on page 381.

10.2.8 Lock user function block class

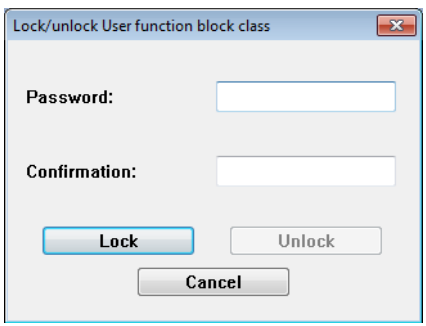
It is possible to lock the implementation of a user function block class with a password.

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 can be commissioned.

A locked user function block cannot be modified.

> Select user function block class in project tree

> **Options > Lock/Unlock UFB Class**



th015us.png

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 dialog 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.

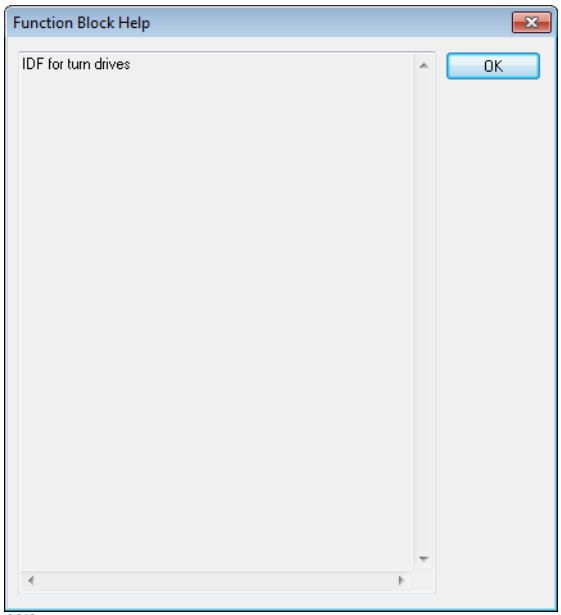
10.2.9 Help for user function blocks

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 a comment.

> Select user function block class in the project tree.

> **Project > Comment**

The help text is called up via the HELP button in the instance parameter dialog of the user function block.



th012us.png

10.2.10 Export and import

A complete user function block class, or certain elements of it, can be exported or imported.

> Select node(s) in project tree > **Edit** > **Block export...**

or

> **Edit** > **Block import...** > select file

10.3 Commissioning

10.3.1Load 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 Freelance operator stations together with the project. This means, the Freelance Operations part of the function block class (faceplate) should be defined in those language which is used for Freelance Operations.

10.3.2 Read, write and correct

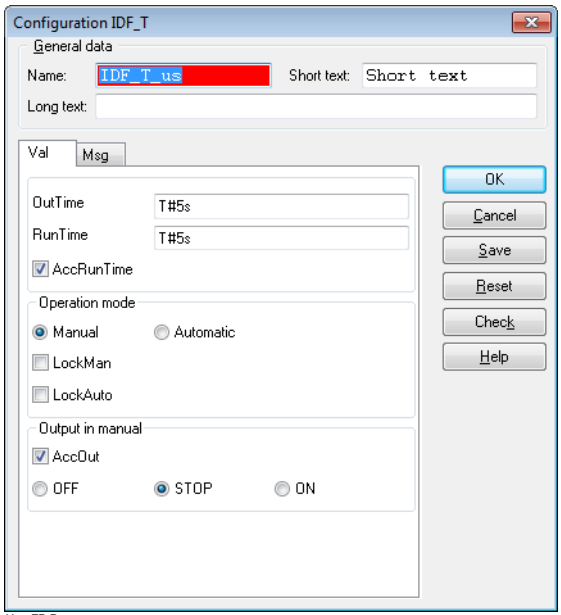
When reading, writing and correcting exported parameters (**PARA_EXP**), the action on the referenced parameters will be displayed for the nested function block. This display can under certain circumstances encompass many layers of nesting.

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.



UserFB Parameter_us.png

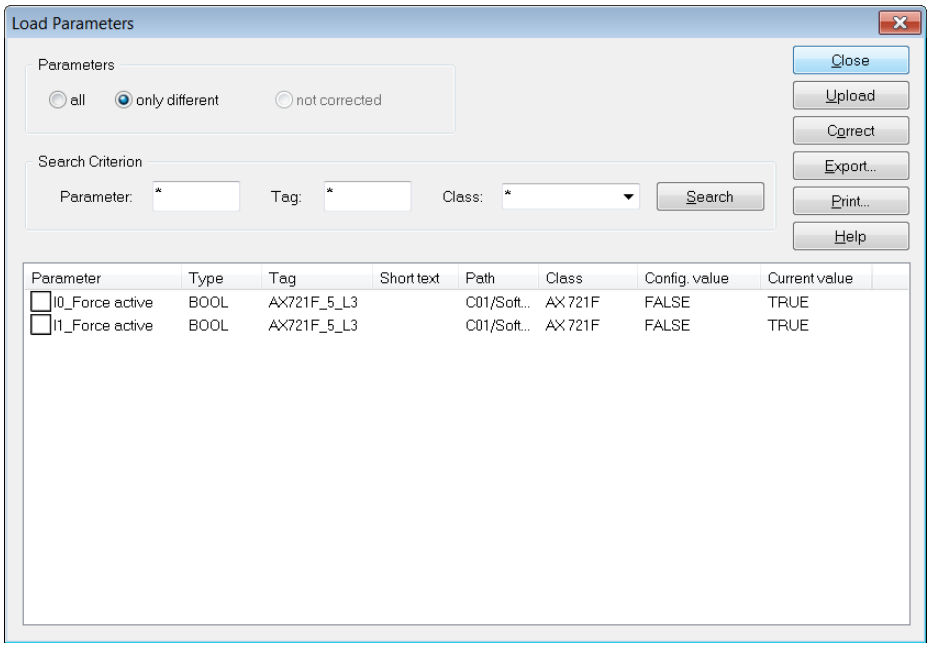
If variables of a user function block instance are being displayed in a value window or a trend window, these values will not be saved when commissioning mode is quit. When commissioning mode is reentered, the values previously in the value or trend window will no longer be available.

10.3.3 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.

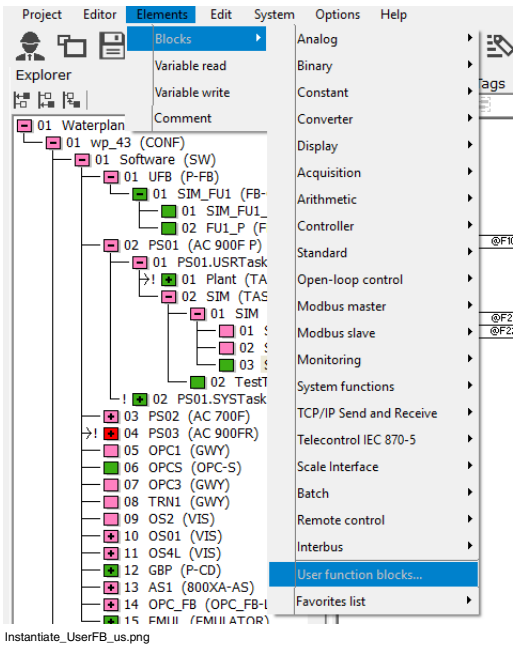
If an exported parameter has a tag name as its source, then the parameter will only be accepted in the most outside point of use in the upload list. This will cause the parameter to be missing from all nested function blocks with tag names.


Load Parameters
Parameters
all
only different
not corrected
Search Criterion
Parameter:
*
Tag:
*
Class:
*
Search
Close
Upload
Correct
Export...
Print...
Help
Parameter
Type
Tag
Short text
Path
Class
Config. value
Current value
I0_Force active
BOOL
AX721F_5_L3
C01/Soft...
AX 721F
FALSE
TRUE
I1_Force active
BOOL
AX721F_5_L3
C01/Soft...
AX 721F
FALSE
TRUE
th007us.png

10.4 Generate instances of user function blocks

10.4.1 Create new user function block instance

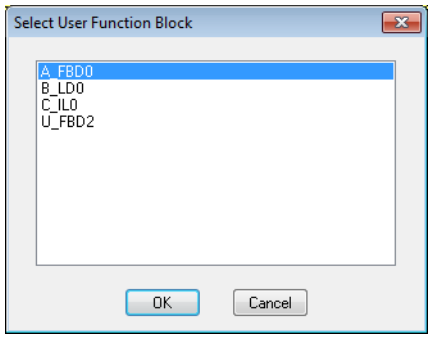
A function block instance is created by first choosing a class out of a list of user function block classes. Instances can only be created from classes which have passed the plausibility check.



Instantiate_UserFB_us.png

> Elements > Blocks > User function blocks

All defined function blocks which have passed the plausibility check appear in the *Select user function block* window. Select and position the desired function block. Alternatively, the user-defined function block class can be selected from the Library pane or via context menu **Blocks** of the graphic area.



di0290us.png

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 Modification of user function blocks on page 393.

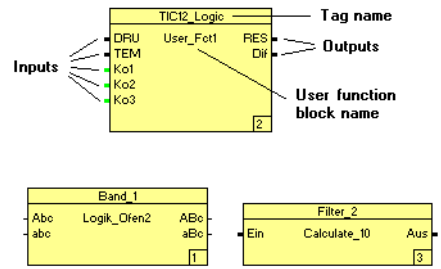
10.4.2 Using user function blocks

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.

Examples of user function blocks



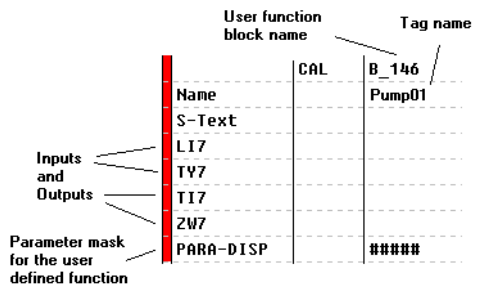
di0288uk.bmp

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.

Example of an user function block



di0277uk.bmp

ST-Program

In ST programs a UFB must be declared as a standard function block. Calling of the UFB takes place in an instruction. Inputs and outputs can also receive and transfer values independently of the UFB. The first three characters of the UFB inputs and outputs are available for the inputs and outputs.

```
VAR
  TI201_LIN: LIN2;
END_VAR

TI201_LIN.X := in1;
TI201_LIN.Y := in2;
out1 := TI201_LIN.Z;
```

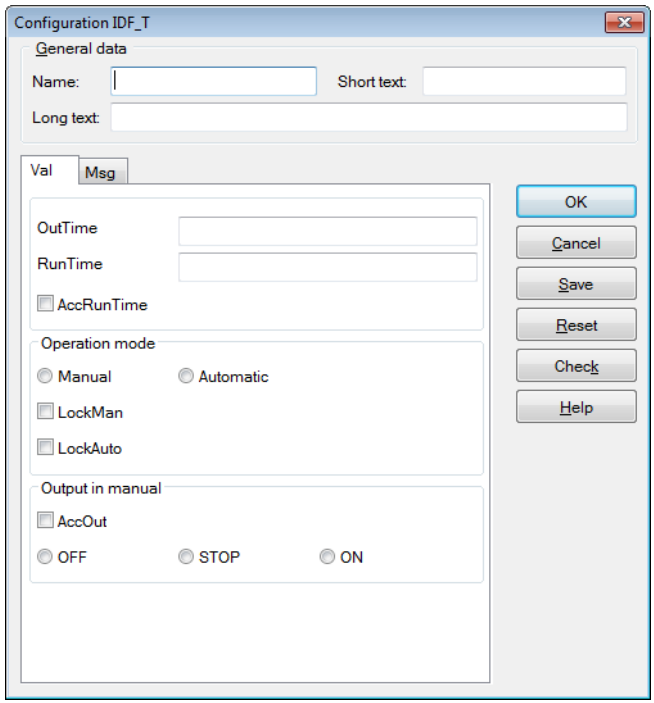
Modify parameter data

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.



ConfDialog_UFB_us.png

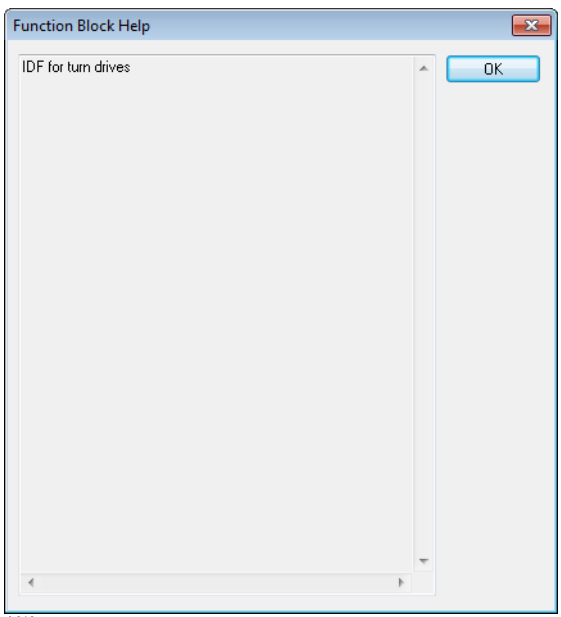
General data

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 allowed.
Long text	Up to 30 characters, all characters are allowed.
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.

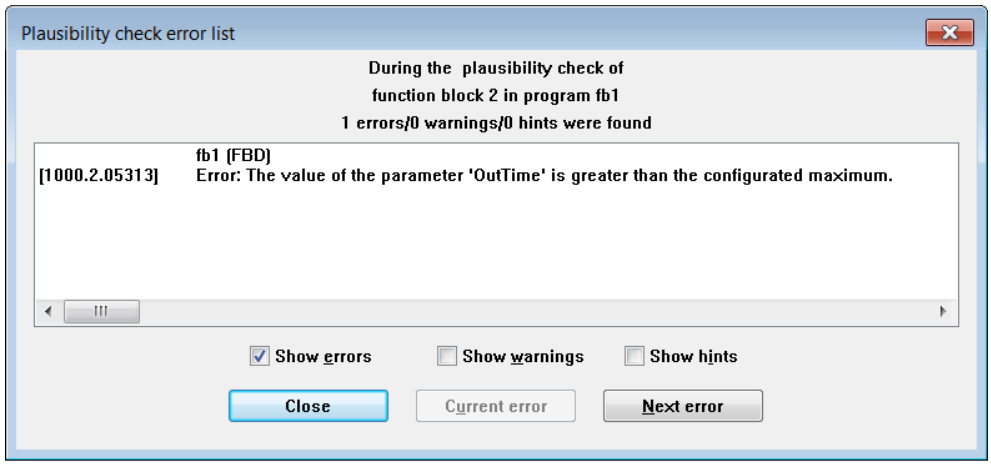
Help Help is provided for the user function block. The comment text associated with the user function block class is displayed as help text.



th012us.png

Check of instances

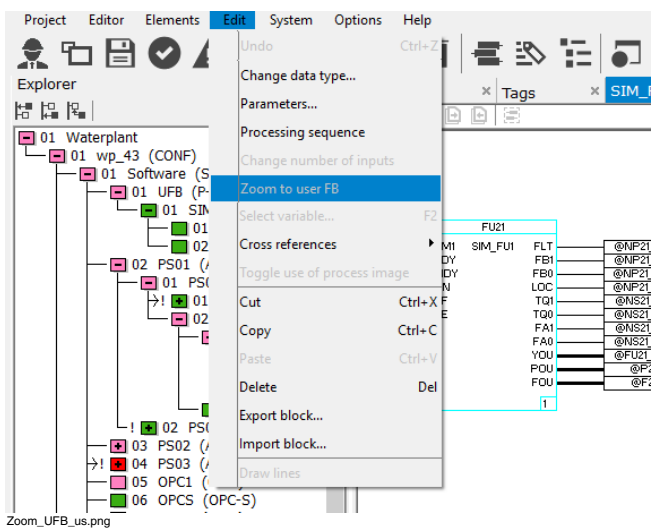
When parameters are entered for a user function block instance, they are checked against any value ranges previously entered in the interface editor. If any ranges are exceeded, the user function block instance is marked as implausible.



th016us.png

Zoom to user function block

Entries in the parameter dialogs 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.



Zoom_UFB_us.png

> Select user-defined function block symbol with single cursor click

> **Edit > Zoom to user FB**

The user function block program is displayed

> Double- click the desired block

> Make required changes in the parameter dialog 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.

10.4.3 Use faceplates of user function blocks

The group display editor can handle any rectangles within a 120-square (30 or 36 fields width, 4 fields height). 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.

10.5 Modification of user function blocks

Changes to UFBs may only be made in the **User function block pool**, that is 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, all instances of the changed user function block class 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).

Add or delete inputs in the user function block

When adding or deleting inputs or outputs, this UFB must be replaced in all the programs used. The parameterized data are lost in the process. The altered UFB, i.e. in this case the instances in the programs are marked 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 program and the faceplate is required. If the change had an effect on the faceplate, then the faceplate must be loaded to the Freelance operator stations.

Changes to the UFB text list

After a change to the text list of a user function block, a plausibility check of the associated program and the faceplate is required. If the change had an effect on the faceplate, then the faceplate must be loaded to the Freelance operator stations.

Changes made in commissioning

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 Freelance operator stations.

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.

Overview of changes and effects

The following table gives an overview of changes done in the class definition and the effects to the class itself and the associated instances:

Effects of changes of a user function block class

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 project 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.	If no substantial change was made. If a substantial change was made.
Change sequence of classes in UFB pool	1. b.	The configured values of the included blocks are preserved; the dependence of the included classes must be considered
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 values of the instances are 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.	
Add variables with storage type VAR_DPS or VAR_VIS	1. b.	

Change	Effect	Comment
Delete variables with storage type VAR_DPS or 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 part are lost.
Rename program node in project tree	2. a.	
Delete program node 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

		Effect			
Change of	**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.

(1) The effect of the change depends on the validity of the initial value and the configured values in the new value range.

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	Instance must be deleted and newly inserted (default values are used)
e.	yes	no	no	yes; values must be possibly adapted

11 Debugger

11.1 General description – Debugger

The debugger is a source text debugger for programming languages to IEC 61131-3. At present only the debugging of structured text programs is supported. The debugger supports all available types of process station.

The debugger is used for tracing faults in programs and should not be used while plant is running.

The most important element of working in the debugger is the breakpoint list. In addition, expressions from the program can be observed in the watch window.

The debugger can only be started in commissioning mode.

11.1.1 Fault tracing with the debugger

Debugging is the discovery and correction of errors ("bugs") in programs. The process can be very time-consuming.

Debugging is not an exact science. The task can be made easier by having a systematic method of work.

The process of fault tracing can be broken down into four main steps.

1. Detecting an error
2. Locating the error
3. Finding its cause.
4. Correcting the error

Is there really an error?

This point may be very obvious. The user task state changes to **not executable**. An output does not assume the expected value. The plausibility check reports an error. Often, however, problems are not so easy to detect. The program may work without errors until a variable takes on a certain value, e.g. zero or a negative number.

Where is the error?

This is often the most difficult step. Looking through a complex program is not easy. For this reason, instead of writing one large program, a number of small programs should be written.

What kind of error is it?

Once the error has been pinpointed it is usually easier to discover why the program is throwing up problems.

How can the error be eliminated?

The final step is to correct the error.

This four-stage process is run through many times when creating a program. For example, the plausibility check reports a lot of syntax errors. The machine code for the program cannot be generated until these have been corrected. The debugger can only be used after the program has been loaded onto the process station.

11.1.2 Breakpoints

A breakpoint is the point at which user task processing is suspended.

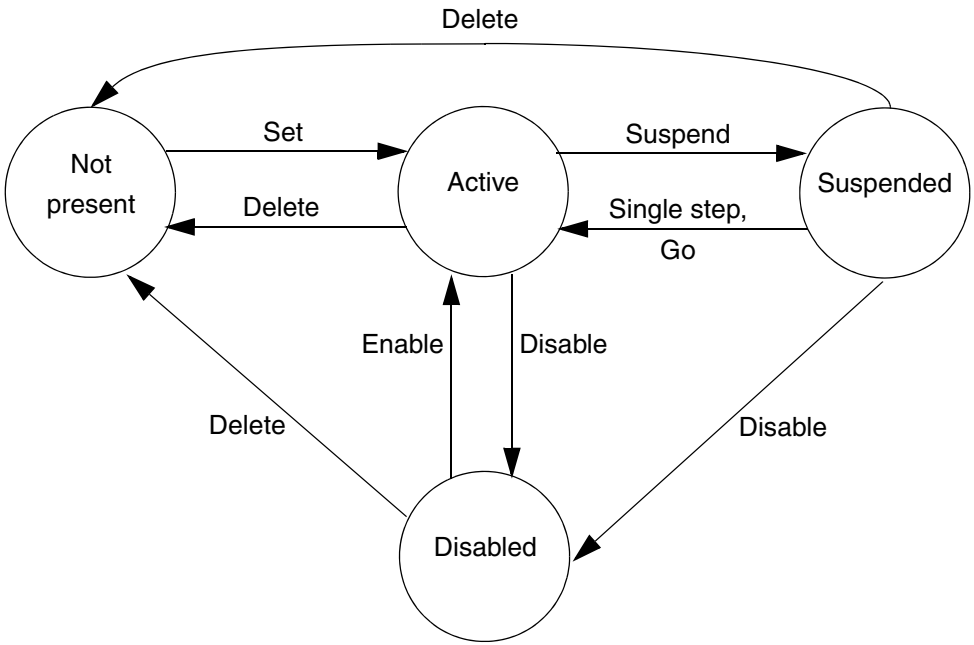
A breakpoint must not be confused with the stopping of the user task. When it is stopped, the user task comes to an end before its status changes to stop. At a breakpoint the user task remains suspended without proceeding further and process image variables are not output.

Breakpoints have the following states:

Not present	The breakpoint is not set.
Active ●	The breakpoint is set and enabled.

Disabled	The breakpoint is set and disabled. The user task is not suspended at the breakpoint.
Suspended	The user task has been suspended at an active breakpoint.This state can only be achieved in commissioning mode.

The following transitional states exist between the individual states:



Set	The user sets a breakpoint in the ST program editor. See Set/delete breakpoints on page 409.
Activate	The user enables a breakpoint in the ST program or in the breakpoint list. See Enable/disable breakpoint on page 409.
Disable	The user disables a breakpoint in the ST program or in the breakpoint list. See Enable/disable breakpoint on page 409.
Suspend	The user task runs through an active breakpoint and is suspended. See Task state on page 409.

Single step, Go	The user continues program processing in single steps (see Single step on page 411) or continuously (see Go on page 412).
Delete	The user deletes a breakpoint in the ST program or in the breakpoint list. See Set/delete breakpoints on page 409.

Breakpoints can be used simultaneously on a number of process stations in the same project. A maximum of 32 breakpoints can be set in a project.

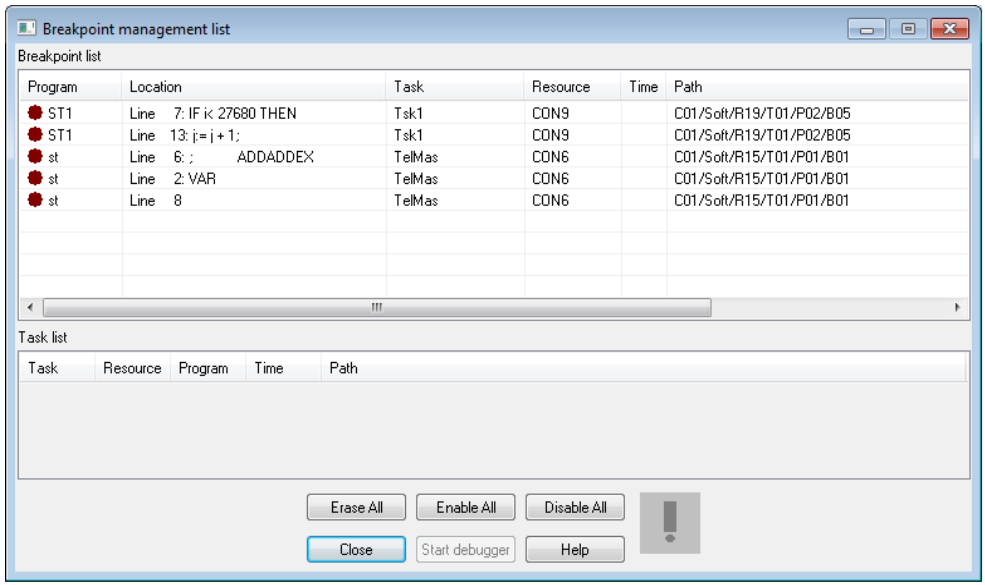
11.2 Debugger interface

11.2.1 Breakpoint list

The list of breakpoints can be called up from the project tree, all program editors, the hardware structure, the variable list and the tag list as well as from the editor for structured data types.

> **System > Breakpoint list**

After calling up the breakpoint list the breakpoint management list is displayed.



tf001us.png

Breakpoint list

The breakpoint list shows all the breakpoints (enabled and disabled) that have been set in the project.

Program Name of the program in which the breakpoint is set.

Location Number of the line at which the breakpoint is set. The first characters of program code are also shown.

Task Name of the task in which the breakpoint is set.

Resource Name of the resource in which the breakpoint is set.

Time Time at which the program jumped to the breakpoint. The field is empty if the breakpoint has not yet been reached.

Path Displays the assignment of the program to the project, the resource, the task and the program list. This can be displayed as **long text or short text**. The setting for this is in the project tree, under **Options**.

Task list The task list shows all the user tasks that have the status **braked**. It is only active if the debugger has been started. The columns are explained in Breakpoint list on page 402.

Erase all Deletes all the breakpoints that have been set.

Enable all Enables all the breakpoints that have been set.

Disable all Disables all the breakpoints that have been set.

Close Closes the breakpoint management list. This does not stop the debugger.

Start/Stop Debugger
Starts and Stops the debugger.

Help Calls up the debugger's online help.

The user can jump directly from the breakpoint list to the breakpoint definition point.

> Select program > context menu > **Jump to source location**

or

> Double-click program in the list

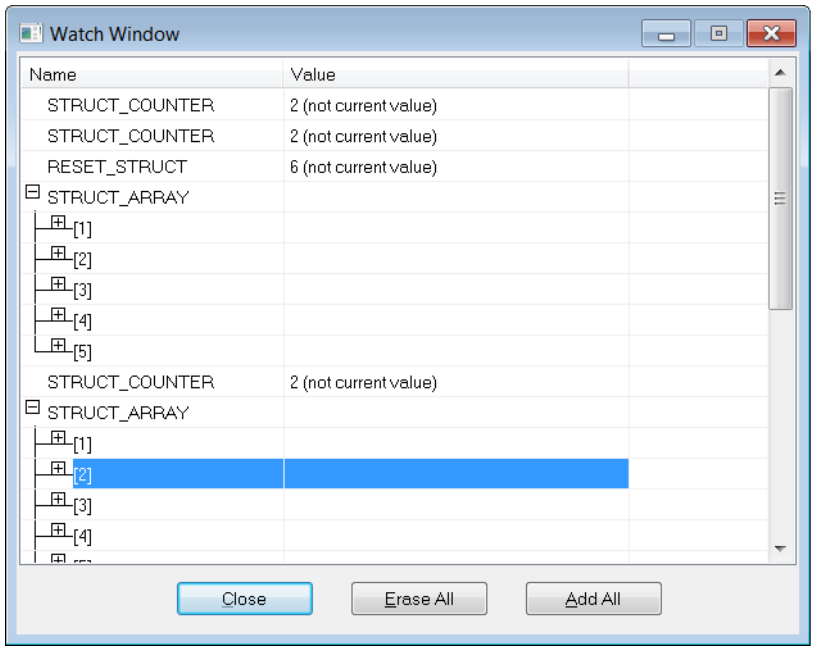
11.2.2 Watch window

The watch window can be used to display the current values of local variables and expressions. The watch window can be called from ST programs.

> **Window > Show watch window**

For variables with a array data type, the individual elements can be displayed. Global variables can only be displayed in the watch window if they are read or written via the process image. The values in the watch window are not updated on each cycle. They are updated

- after opening the watch window and
- after every single step.



tf002us.png

Name Name of the variable or expression to be evaluated.

Value Value of the variable or expression.

A valid value is only displayed if the state of the associated user task is **braked**. Otherwise the fact that the value is not valid is shown by *not current value*.

If a value changes after a single step, this value is shown in red.

Close Closes the watch window. All the variables and expressions entered are saved in the project and are displayed again the next time they are called.

Delete all Deletes all the entries in the watch window.

Display all Appends all the local variables to the entries that are present in the watch window.

Add watch entry

> Context menu > **Insert**

or

> **INS**

The name of a variable or an expression can be entered.

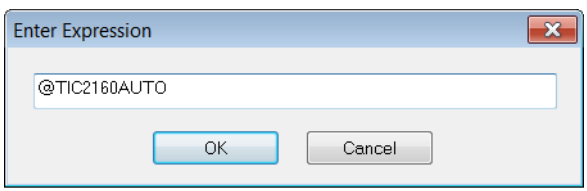
An expression can contain any number of variables, numeric constants and operators. Functions cannot be used. The data types of variables and constants used in an expression must be compatible with each other.

Individual elements of structured variables and array variables can also be added, e.g.

Variable_name.component_name
Variable_name[3, 1, 1, 7]
Variable_name[3, 1].component_name
*Variable_name[2 * index + 1]*

In the expression for calculating the index a maximum of one variable can be used and only the operators +, -, * and /.

Global variables can only be displayed if they are read or written via the process image. The @ must also be quoted, e.g.



tf009us.png

Write value

> Mark entry > context menu > **Write**

Values can only be written for variables. The value entered must be of the same data type as the variable. The variable contains the written value until the program or the user assigns a new value to it.

A value can only be written if the state of the associated user task is **braked**.

The value of an expression can only be altered by changing the value of the individual variables.

Change a watch entry

> Mark entry > context menu > **Edit**

or

> Double-click an entry

The selected name of the variables or expression can be altered.

Delete a watch entry

> Mark entry > context menu > **Delete**

or

> **DEL**

The marked block in the watch window is deleted.

11.3 Working with the debugger

11.3.1 Starting the debugger

When the debugger is started, the breakpoints that are already set are loaded onto the process station. The debugger can only be started in commissioning mode.

> System > Breakpoint list > Start debugger

The state (Debugger active) is displayed in the title line. If a breakpoint is set once the debugger has started, the task is set to suspend when it passes through the breakpoint without requiring confirmation.

11.3.2 Edit breakpoint

In ST programs, breakpoints can only be set in program lines with executable statements. A maximum of 32 breakpoints can be set in the whole project.

```
FOR Index:=1 TO 10 DO
  IF @IN <= Curve[Index,1] THEN      (* Sector found *)
    IF Index = 1 THEN               (* low limit *)
      @OUT := Curve[Index,2];
      EXIT;                    (* Exit FOR loop *)
    END_IF;
(* Calculation of output according to the straight line equation *)
(* Y := (((Y2 - Y1) / (X2 - X1)) * (X - X1)) + Y1; *)
    OUT := (((Curve[Index,2] - Curve[Index-1,2]) /
             (Curve[Index,1] - Curve[Index-1,1])) *
            (IN -Curve[Index-1,1])) + Curve[Index-1,2];
    EXIT;                    (* Exit FOR loop *)
  ELSE
    IF Index = 10 THEN              (* high limit *)
      @OUT := Curve[Index,2];
      EXIT;
    END_IF;
  END_IF;
END_FOR;
```

tf003us.bmp

Breakpoints can also be set in programs that are computed in the system tasks. As system tasks are not computed cyclically, suspension at the breakpoint is associated with the event concerned.

Cold start task	With a cold start, the connection between Freelance Engineering and the process station is broken. This disables all active breakpoints on the process station. They can only be enabled once the cold start has been completed.
Warm start task	With a warm start, the connection between Freelance Engineering and the process station is broken. This disables all active breakpoints on the process station. They can only be enabled once the warm start has been completed.
Run task	Run task is processed when the resource starts. The program is suspended at active breakpoints in the flow.
Stop task	Stop task is processed when the resource stops. The program is suspended at active breakpoints in the flow.
Error task	The error task is processed when an error occurs in a user task. The program is suspended at active breakpoints in the flow.
Lateral tasks	No programs can be configured under lateral tasks.

Set/delete breakpoints

In ST programs, breakpoints can be switched on and off. If there is no breakpoint in the line the switch sets it, if there is one it deletes it.

Position the cursor on the desired line
In configuration mode: > **Edit** > **Toggle breakpoint**
In commissioning mode: > **Debug** > **Toggle breakpoint**.
or
F9

In the breakpoint management list it is only possible to delete breakpoints.

Mark block in the breakpoint list

> Context menu > **Delete**

or

> **DEL**

Enable/disable breakpoint

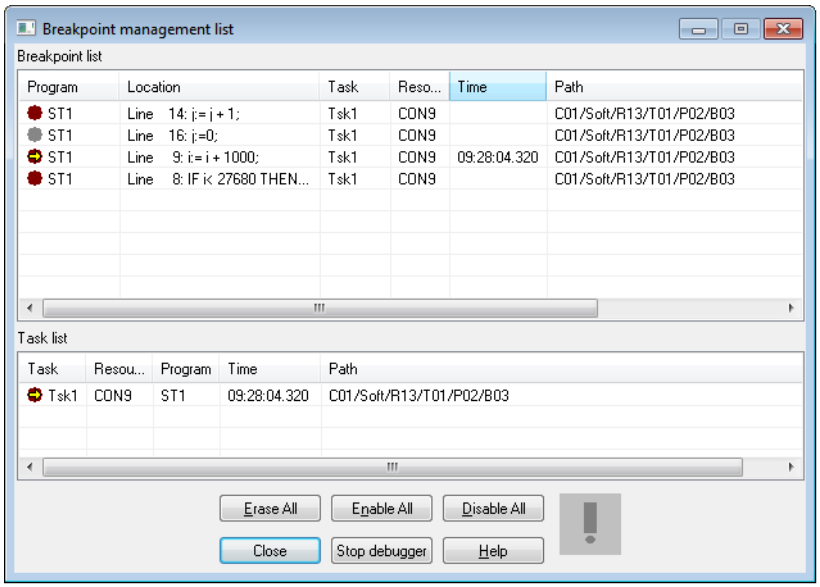
Breakpoints that have been set can be disabled and enabled. Disabled breakpoints are displayed in grey. Breakpoints can be enabled and disabled in both the ST program and in the breakpoint management list.

Position the cursor on the desired line
In configuration mode: > Context menu > **Disable / Enable breakpoint**
In commissioning mode: > **Debug** > **Disable / Enable breakpoint**

Mark block in the breakpoint list > Context menu > **Enable / Disable**

11.3.3 Task state

If an active breakpoint is passed once the debugger has started, the user task is suspended in front of the statement at which the breakpoint is located and its status is set to **braked**. The breakpoint processing dialog opens. The state **braked** is displayed in the project tree and in the user task dialog.



tf004us.png

In the breakpoint management list the suspended programs (breakpoint list) and user tasks (task list) are identified by a yellow arrow (Program cursor) in the breakpoint symbol. The program cursor is displayed all the time that the state of the user task is **braked**.

The user can navigate directly from breakpoint management to the suspended programs.

Select program or user task > Context menu > **Jump to Source Location**

or

> Double-click program or task

While the state of a user task is **braked**, no variables can be updated via the process image. All user tasks that do not have **braked** as their state are processed normally.

If the resource is **stopped** while a user task is **braked** the task state changes to **not executable** with the error *Invalid command in break*. The error task is therefore **not** started.

If the resource is **stopped** while a program in a user task is in an endless loop the task state changes to **not executable** with the error *Execution aboard*. The error task is therefore **not** started.

If, while the debugger is running, the connection between Freelance Engineering and a process station is broken, the breakpoints of this process station are set to disabled in the breakpoint list. The broken connection is indicated by a flash symbol in the breakpoint list. The breakpoints loaded on the process station are deleted.

After the connection has been re-established the breakpoints must be enabled once more. This is done by reloading them onto the process station.

11.3.4 Single step

Single steps are only possible in programs in which a breakpoint has been reached.

> **Debug > Single step**

or

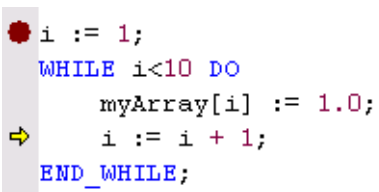
> **F10** key

The single step enables the program to run one step at a time. Every single step processes the statement which is marked by the program cursor (yellow arrow, see also the picture below). The program cursor then moves to the next statement to be carried out.

For keyboard operation (**F10**), the focus must be on the ST program.

While the program is running in single steps the state of the user task remains **braked**.

If the ending condition is not satisfied at the end of a loop the program cursor is placed at the beginning of the loop. In conditional loops the program cursor only runs through the condition that is satisfied.



tf005.bmp

Statements that do not occupy exactly one program line are special cases.

- More than one statement on a line
  The execution stops before the first statement. After a single step all the statements in the program code line are carried out. The program cursor goes to the next statement in a new line.
- One statement on more than one line
  The program cursor is positioned in the last line of the statement. The statement is carried out with a single step and the program cursor goes to the next statement in a new line.

After the last statement in a program, it is not possible to change to the next program with a single step. In this case program processing is continued.

11.3.5 Watch values

The values are updated in the watch window at every single step. Changes to the displayed value are shown in red.

On going through the text of the ST program the current values of the variables are displayed.

11.3.6 Go

After a breakpoint or single steps in the program have been enabled, program processing can Go. The user task then leaves the **braked** state.

> **Debug > Go**

or

> **F12** key

11.3.7 Stop debugger

System > Breakpoint list > Stop debugger

The debugger is terminated. The breakpoints loaded onto the process station are deleted and cyclic program processing of the user task continues.

The debugger is terminated automatically on changing to configuration mode. The breakpoints retain their state in the breakpoint list but are deleted on the process stations.

11.3.8 Typical examples of errors

Examples of typical errors found during fault tracing using the debugger are explained below.

Endless loop

Endless loops are normally difficult to find. The following program should initialize the elements of a array.

```
PROGRAM init
VAR
  i: DINT := 1;
  myArray: ARRAY [1 .. 10] OF REAL;
END_VAR

FOR i:=1 TO 10 DO
  myArray[i] := 1.0;
  i := i - 1;
END_FOR;

END_PROGRAM
```

1. Detecting the error
   Since the user task is in an endless loop it does not calculate or output any new values
   When a user task is in an endless loop the following behavior occurs:
   - The CPU load is 100% even without a default task.
   - It is not possible to load changes. The event log reports the error: E_DMS_INSTALL_TIMEOUT. See Event log on page 420.
   - Breakpoints that have been set outside the endless loop are not enabled. This also applies to other user tasks with the same priority.
   - When the task is stopped, an error message is issued after a delay: *Timeout occurred, service did not respond.*

– When the resource stops there is a waiting time, then the state of the user task concerned changes to **not executable** with the error *Execution abort*.

2. Locating the error

   If all the indicators point to an endless loop this does not make it easier to locate. The endless loop is probably in an ST program. However, they can also be found in IL and LD programs.

   Location can be helped by retracing the last changes. If the changes were extensive, it is a good idea to proceed step by step. The error can be given limits by stopping all project tree objects and starting individually. After the affected user task has been found, continue with the individual program lists. The breakpoints can now be used to analyze each ST program individually.

3. Finding the cause

   The causes of endless loops are many and varied. In every case, the ending condition of the loop is not reached. The end condition should therefore be analyzed using the debugger and the watch window.

   In the present example the controlled variable within the loop is repeatedly decremented. This causes the loop index to remain constant and therefore the ending condition is not reached.

4. Correcting the error
   To satisfy the ending condition of the loop, the unwanted decrementation of the controlled variable is deleted:

```
FOR i := 1 TO 10 DO
  myArray[i] := 1.0;
END_FOR;
```

Overflow

In the following sample program, initialization has been changed from a **FOR** loop to a **REPEAT** loop.

```
PROGRAM init
VAR
  i: DINT;
  myArray: ARRAY [1 .. 10] OF REAL;
END_VAR
```

```
i := 1;
REPEAT
  i := i + 1;
  myArray[i] := 1.0;
UNTIL i>10 END_REPEAT;

END_PROGRAM
```

1. Detecting the error
   In this case the error is easy to detect. After loading the program the user task state changes to **not executable**. In the user task dialog the error displayed is *Illegal array index*.
2. Locating the error
   Location as far as the ST program affected is simple. The user task dialog displays the error object. Use **Info** to find the path to the ST program that is causing the error.
   If necessary the breakpoints should be used to narrow down the site of the error in the ST program.
3. Finding the cause
   Access to array elements outside the defined array area is not allowed because this also accesses storage areas that do not belong to the array.
   In the present example the ending condition tests for `i > 10`. `i = 10` does not satisfy the condition yet and the loop recommences. The index increases to `i := 11`. However, the array is only defined in the range `[1 .. 10]` and the current index references an element that is outside the array.
4. Correcting the error
   The index must be limited to the value 10, e.g.
   `UNTIL i>=10 END_REPEAT;`

Loop counters

The above program has been corrected. After initialization the sum is determined by the array objects.

```
PROGRAM init
VAR
  i: DINT;
```

```
  myArray: ARRAY [1 .. 10] OF REAL;
  sum: REAL;
END_VAR

i := 1;
REPEAT
  i := i + 1;
  myArray[i] := 1.0;
UNTIL i>=10 END_REPEAT;

sum := 0.0;
FOR i := 1 TO 10 DO
  sum := sum + myArray[i];
END_FOR;

END_PROGRAM
```

1. Detecting the error
   The error in the program is difficult to find because the user task runs error-free.
   Checking the value of the variable `sum` gives the result 9. The sum of a array with 10 objects, all initialized at 1, should be 10.
2. Locating the error
   An error of this kind is normally only found later in the signal path when an output takes on an unexpected value. Tracing the signal path back by means of cross references enables the faulty program to be arrived at.
3. Finding the cause
   Checking the array elements in the watch window shows that the first array element was not initialized.
4. Correcting the error
   The index is initialized at 1 and already incremented before the first array access. Thus the first initialization is with the array element 2. Array element 1 is bypassed. In the loop only 9 array elements are initialized.
   If the index is initialized at 0, all 10 array elements are initialized.

Other kinds of error

Initialization

If, for example, the sum is to be recalculated by a array in every task cycle, initialization at the variable declaration will prove unsuccessful.

```
VAR
  sum: REAL := 0.0;
END_VAR

FOR i := 1 TO 10 DO
  sum := sum + myArray[i];
END_FOR;
```

Since the local variable `sum` is only initialized on loading the program and not every time it runs, it is the task cycles that calculate the sum. The variable `sum` must be initialized every time before the sum is calculated:

```
sum := 0.0;
FOR i := 1 TO 10 DO
  sum := sum + myArray[i];
END_FOR;
```

Process image

Access to global variables via the process image is normally recommended. However, within loops this can lead to problems as the process image is only updated at the beginning and end of calculation of a user task. The following example leads to an endless loop:

```
VAR
  StartTime: DT;
  TimeDiff: TIME;
END_VAR

StartTime := @ps12.DateTime;
REPEAT
  TimeDiff := SUB(@ps12.DateTime, StartTime);
UNTIL TimeDiff > t#2ms END_REPEAT;
```

Since the process image is not updated during the running time of the task the time difference `TimeDiff` remains at 0 and the loop does not end. In this case the system variable must be accessed directly (without the process image).

Decrementation in loops

When data types with no sign are decremented there should be no test for 0.

```
VAR
  w: UINT;
END_VAR

w := 5;
WHILE w >= 0 DO
  w := w - 1;
END_WHILE;
```

After the fifth iteration `w` equals 0. The next time it is 65535 (value range of the UINT data type), which is still greater than 0. This loop therefore never ends.

11.4 Breakpoint functions

11.4.1 Mark breakpoints

Mark an individual breakpoint

> Place cursor on desired breakpoint > click left mouse button

The whole line of the breakpoint is the selection area.

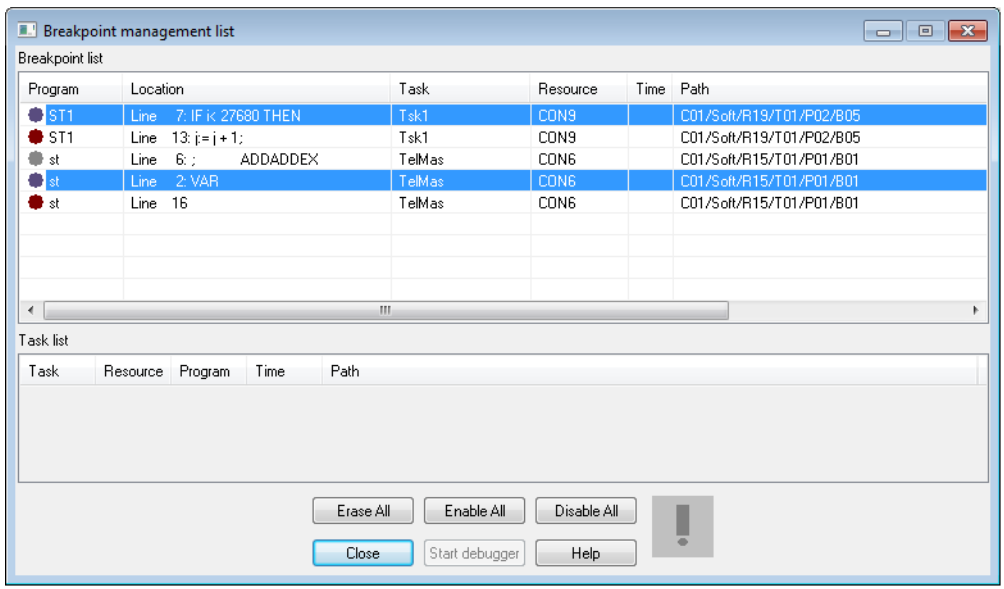
Mark a number of breakpoints

> Hold down **SHIFT** key > click left mouse button to mark breakpoints

Inside a line, characters are marked. Otherwise complete lines are marked. After marking, the desired operation can now be carried out. Example: > **Disable**.

Mark additional breakpoints

> Hold down **CTRL** key > click left mouse button to mark additional breakpoints



tf006us.png

The additional characters or lines are included in the marked area.

Jump to a breakpoints

> Context menu > **Jump to Source Location**

or

> Double-click the breakpoint

The program in which the breakpoint is defined opens with the cursor positioned in the line with the breakpoint.

Deselect breakpoints

> Click an unmarked breakpoint

A selection is automatically canceled by closing breakpoint management.

Save breakpoints

Breakpoints are saved with their state in the project. They are not exported with it.

Save the watch window

The settings of the watch window with all the variables and expressions are stored for each program in the project. They are not exported with it.

11.4.2 Event log

All loading operations while Freelance Engineering is running are logged in the Windows event log. The relevant entries are contained in the application log.To display the event log, the event display must be started.

The event log has three categories which can be filtered.

- Information
- Warnings
- Errors

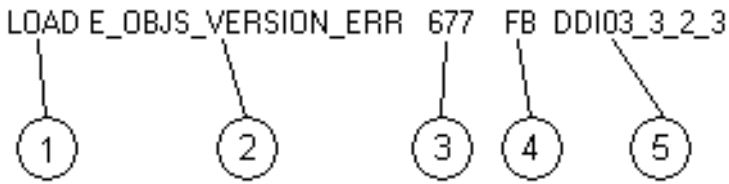
A detail view can be called up for every entry in the event log. Loading errors during commissioning are shown by the message box

```
At least one loading process has failed
```

and are recorded in the event log as errors. The individual entries have the following structure:

```
LOAD E_OBJ S_NO_MEM 819 PRG Program
fwk2/conf/ps_1/ps_1.USRTask/Task21/RecWrType_s
NOLOCK
```

The first line gives information on the actions carried out, the error that occurred and the object that is affected by the error.



dg004.bmp

1. Action
   Action that has caused the error.

PARA-WRITE	Error due to a Freelance Engineering write access
PARA-CORRECT	An error occurred during correction
LOAD	Error while loading the project

2. Error
   Displays the error that has occurred.

3. Object number
   Number, in the object directory, of the object which has caused the error.

4. Classification of the object
   Type of the object that has caused the error.

FB	Function block
TSK	Task
PRG	Program
CLS	Function block class
RED	Redundancy object
PI	Process image

5. Object name
   Name of the object that has caused the error (e.g. tag name for function blocks).

A second line shows the path of the object in the project tree.

The last line always contains the user reported in SecurityLock or NOLOCK if SecurityLock is not installed.

Examples of errors that have been recorded in the event log:

Load errors

```
LOAD E_DMS_NO_MEM 554 CLS C_CU
fwk2/conf/ps_1
NOLOCK
```

The function block class `C_CU` (Universal controller, continuous) could not be loaded onto resource `ps_1`. Insufficient memory is available on resource `ps_1`.

To remedy the error, the free storage space on the resource can be analyzed (system variables of resource) and subsequently the PRAM store (boot parameters of the resource) can be increased.

Write errors

```
PARA-WRITE FAIL binab1 M_BOUT Parameter:Phz BOOL
doku_vis_v42/conf/D_PS/D_PS.USRTask/Main/Bst/ANAUE
NOLOCK
```

An error occurred while parameter `Phz` was being written in function block `binab1` (Function block class M_BOUT - binary output assignment).

The block showed the value because in the current operating mode it was not possible to write the parameter `Phz`. After changing the operating mode the parameter `Phz` can be successfully written.

List of selected errors

E_DMS_INSTALL_TIMEOUT	Timeout during loading. Cause: an endless loop is configured in a user task. Remedy: Correct the program.
E_OBJS_NO_MEM	Cause: there was not enough free space to load the object in one of the storage areas. Remedy: Optimize storage allocation.

E_DMS_STATION_ABORT	Cause: the connection between the system bus and the station was interrupted during the loading process. Remedy: Repeat the loading process.
E_OBJS_VERSION_ERR	Cause: the version number of the instance to be loaded does not agree with the loaded class. Remedy: Load the current class (Project - plausibility check all, load whole station).

Index

A

Accumulator (IL program) 169
ARRAY 240
Array 240
Array index 241
Assign tag 324, 327
Assignment 247
Auto Router 121, 193

B

Bookmark 279
Break point 280
Breakpoint 400, 408
    Enable/disable 409
    Set/delete 409
Breakpoint list 402

C

Call IL operators 168
CASE statement 252
Class 94, 353
Coils (LD) 199
Comment 24, 238
Commissioning
    Instruction list 185
    Ladder diagram 224
    Sequential function chart (SFC) 340
    Structured text (ST) 291
Conditional statement 250
Connections (LD) 196
CONST 236
Contact plan
    Elements 199
Cross reference 287
Cross references
    Tag list 75
CSV (comma separated values). 59

D

Data Types (Variable list) 22
Debugger 399
    Exit 412
    Single step 411
    Start 407
    Task state 409
Delete 36
Display of steps 347, 349

E

Edit field 34
Edit list entries 31
Event log 420
Example (SFC) 296
Example of a transition program (SFC) 321
EXIT statement 256
Export
    Tag list 72
Export and import blocks (SFC) 337
Export block 337
Export flag 24

F

Faceplate 393
FOR statement 253
Function
    Data type 271

Input number 272
Use 270
Function block 244, 249, 272
Check 276
Insert 266
Parameterize 275
Pin 273
Function block type
Tag list 61

G

Global variable 241
Global variables 46

H

Horizontal connection (LD) 197
Horizontal sequence selection line SFC 304
Horizontal simultaneous sequence line (SFC) 306

I

Identifier 236
IF statement 250
IL operators 165
IL program
Create 158
Import block 337
Initial step SFC 302
Initial value 25, 55
Instance 94, 353
Instantiate 267

J

Jump (LD) 204
Jump SFC 303

L

Label (LD) 206
Ladder Diagram 187
LD program
Contact 198
Create 189
Elements 196
Function blocks 203
Library type
Tag list 61
Local variable 242
Long text
Tag list 61
Loop operators (IL) 172
Loops 253

M

MO message 330
Monitoring time 317

N

Normal view_Tag list 65

O

OPC address 25
Operand 245
Operator 227
Operators IL program 168, 176

P

Plausibility state
Tag list 62
Pool 94, 354
PowerFail 51
Process image 25, 33, 45, 266
PROGRAM 238, 277
Program 374
Project
Version number 47

R

Relational operators (IL) 172
Release tag 324, 328

REPEAT statement 255
Replace 281
Resource assignment 43
Return (LD) 204
RETURN statement 257

S

Sequence selection
convergence add SFC 306
convergence end SFC 306
divergence add SFC 305
divergence start SFC 305
Sequential function chart 293
SFC operating mode 331
SFC operating time 331
SFC program
Calling up 295
Create 295
Parameters 330
User interface 298
Shift operators (IL) 172
Simultaneous sequence
convergence add (SFC) 308
convergence end (SFC) 308
divergence add (SFC) 307
divergence start (SFC) 307
Single step 411
ST program
Check 153, 184, 223, 290, 339
Create 228
State of processing 61
Statement 227, 247
Station view
Tag list 65
Step / Transitions operation 333
Step and Transition 342
Step parameters 315
Step SFC 302
Storage type 97, 359
String variables 23
Structured text 227
Call 229
Structured variable 56
System 45
System variable 243, 270
System variables 46

T

Tabulator width 233
Tag list
Area 61
Library type 61
Long text 61
Name 60
Object type 61
Plausibility state 62
Processing stae 61
Processing state 61
Short text 61
Type name 61
Tag name
Tag list 60
Text area
Copy 284
Delete 285
Export 286
Import 286
Mark 283
Paste 285
Write file 285
Transition
state 348
Type 24, 83, 239

U

User Function Blocks 351
Faceplate 393
Modification 393
Storage type 97, 359

V

VAR 242
VAR_EXTERNAL 242
Variable 241
    Access 269
    Insert 265
    Process image 269
Variable (LD) 201
Variable list 21
    Comment 24
    Edit field 34
    Export flag 24
    Initial value 25
    OPC address 25
    Structure 82
    Type 24, 83
Variable name 24, 83
Variables of SFC program 334
Vertical connection (LD) 197

W

Waiting time 317
Watch window 404
WHILE statement 256

Z

Zoom 392

—

www.abb.com/freelance
www.abb.com/controlsystems

—

3BDD012504-111 A